E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Arquitectura Tecnológica del Sistema de Información de la Empresa BDA

Máster Universitario en
Ingeniería Informática

Trabajo Fin de Máster

Christopher Castro Alvarado

Raúl Orduna Urrutia, Carlos López Molina

Pamplona, 18 septiembre 2017

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Trabajo Fin de Máster

Publicación 2.4rc3

Christopher Castro Alvarado

sept. 17, 2017

Índice general

Resumen

El autor de esta memoria en los últimos años ha estado vinculado al quehacer de un bufete de abogados cuya actividad se despliega de manera transversal sobre gran parte del territorio español, y centrada en una serie de servicios tipificables según la naturaleza de los asuntos que conllevan. Esta experiencia le ha permitido reconocer las circunstancias que afectan a este tipo de actividad y algunas necesidades específicas que afectan al sector. En este contexto y frente a tales necesidades, ha debido generar respuestas en orden a desarrollos informáticos que las resuelvan, en las cuales la aplicación de los principios y técnicas para la gestión de proyectos de desarrollo informático, según se han tratado como contenidos centrales del Máster en Ingeniería Informática, han constituido una herramienta clave para enfrentar la tarea con éxito y eficiencia.

Palabras clave: arquitectura de software, gestión de proyectos, abogacía, procesos judiciales, análisis de procesos, sistemas de información.

CAPÍTULO 1

Introducción

La vida profesional reciente del autor de esta memoria transcurre vinculada a una empresa de servicios de abogacía, cumpliendo el rol de “Arquitecto y desarrollador de software”, en esta experiencia le ha correspondido gestionar un proyecto de desarrollo e implantación de un Sistema de Información desde la identificación de necesidades hasta el desarrollo concreto de las soluciones hardware y software adecuadas a los procesos y características particulares de la organización y de su actividad. En este proyecto se ha aplicado las herramientas y metodologías de gestión de proyectos informáticos estándar, reconocidas como buenas prácticas en el sector y que constituyeron temas centrales y claves durante los estudios del Máster de Ingeniería Informática, y en virtud de esta condición se ha recogido de forma natural como Trabajo de Fin de Máster.

Esta memoria recoge el relato de orden histórico de los hechos y situaciones, previas al desarrollo del Trabajo de Fin de Máster, que condujeron a la decisión de desarrollar el proyecto y la herramienta informática a la que dio lugar, y presenta también el recuento de algunos de los aspectos técnicos más relevantes que constituyeron el despliegue del proyecto en sus diversas fases y aspectos claves como de la presentación de resultados apreciables por el usuario. El Trabajo de Fin de Máster que se presenta a través de esta memoria, se ha centrado en la experiencia antes descrita, con foco en las fórmulas aplicadas para determinar y gestionar la realización de los desarrollos informáticos necesarios y orientados a resolver necesidades específicas presentes en la organización de referencia en que han transcurrido las experiencias que se relatan, y que se concretan en un sistema de información asociada a casos jurídicos eficaz, eficiente y adecuado a las condiciones del contexto en que se despliega la actividad de dicha organización.

Concluye con una revisión de resultados en orden global del proyecto, desde la perspectiva de la aplicación de competencias con nivel Máster en Ingeniería Informática, como también de los efectos de tal despliegue desde la perspectiva del diseño de arquitectura del sistema, sobre los resultados de la concreción de los desarrollos de hardware y software del sistema en los niveles siguientes de despliegue del proyecto. Estos resultados se expresan en términos del grado en que se consiguieron los objetivos tanto del proyecto como del trabajo de fin de máster, y se presentan en términos de conclusiones como argumentos que justifican el otorgamiento del grado académico de Máster en Ingeniería Informática para el autor.

CAPÍTULO 2

Aclaración de Términos, Expresiones y Acrónimos

A lo largo de este documento se emplea una amplia gama de términos, expresiones y acrónimos tanto de índole técnica como ligadas al contexto jurídico. En este capítulo se presentan aquellos que se han considerado necesarios para aclarar el sentido semántico con que cada término clave, concepto, expresión, o acrónimo, es aplicado en el ámbito de esta memoria.

2.1 Términos y Expresiones

- **Acoplamiento**: indica el nivel de dependencia entre las unidades de software de un sistema informático, es decir, el grado en que una unidad puede funcionar sin recurrir a otras. Tipos de acoplamientos:
  - **Acoplamiento normal**: una unidad de software llama a otra de un nivel inferior y tan solo intercambian datos. Dentro de este tipo de acoplamiento podemos encontrarnos 3 subtipos, dependiendo de los datos que intercambien las unidades de software. Estos tres subtipos son:
    1. **Acoplamiento de datos**: dos módulos están acoplados por datos si ellos se comunican por parámetro. Siendo un parámetro una unidad elemental de datos.
    2. **Acoplamiento de marca**: dos módulos aparecen acoplados de marca si ellos se refieren a la misma estructura de datos local. Por estructura se entiende un grupo compuesto de datos en vez de argumentos simples.
    3. **Acoplamiento de control**: es cuando un módulo pasa a otro indicadores de control. Provoca dependencia de ejecución entre un módulo y otro.
  - **Acoplamiento externo**: las unidades de software están ligadas a componentes externos, como por ejemplo dispositivos de entrada / salida, protocolos de comunicaciones,

etc.

- **Acoplamiento común**: dos unidades de software acceden a un mismo recurso común, generalmente memoria compartida, una variable global o un fichero.
- **Acoplamiento de contenido**: ocurre cuando una unidad de software necesita acceder a una parte de otra unidad de software.

- **ActiveRecord**: en Ingeniería de software, ActiveRecord es un patrón de arquitectura encontrado en aplicaciones que almacenan sus datos en Bases de datos relacionales.
- **Arquitectura**: de un sistema de información es un diseño que muestra los bloques de construcción físicos y lógicos de una aplicación distribuida (o algún otro sistema de software) y las relaciones entre ellos. En el caso de una aplicación de empresa distribuida, el diseño arquitectónico utiliza generalmente la arquitectura lógica de la aplicación y la arquitectura de implementación.
  - **Arquitectura de datos**: conjunto de reglas, políticas, estándares y modelos que relacionan y definen el tipo de datos tratados y cómo han de utilizarse, almacenar, gestionar e integrar dentro de una organización y sus sistemas de bases de datos. Proporciona una formalización sobre los flujos de creación y gestión de los datos, y sobre cómo han de procesarse a través de los sistemas IT de la organización.
  - **Arquitectura física**: un diseño general que determina la asignación de una arquitectura lógica a un entorno electrónico. El entorno físico incluye los equipos de un entorno de intranet o Internet, los enlaces de red que se establecen entre ellos y otros dispositivos físicos necesarios para la compatibilidad del software
  - **Arquitectura lógica**: un diseño que representa los bloques de construcción de una aplicación distribuida y las relaciones (o interfaces) existentes entre dichos bloques. La arquitectura lógica incluye los componentes de aplicación distribuidos y los componentes de los servicios de infraestructura necesarios para la ejecución de las aplicaciones.
- **BDA**: denominación ficticia con que el autor a designado la organización (real) en la cual se realizó el proyecto que da pie a esta memoria de fin de máster, con el propósito de proteger sus intereses e identidad. Gestiona servicios de abogacía en contexto nacional con más de cincuenta oficinas o delegaciones en toda España, con una plantilla de más de setecientos profesionales, y decenas de miles de clientes también usuarios de sus sistemas informáticos de información vía Internet. Representa el acrónimo “Bufete De Abogados”, que indica el tipo de actividad de la empresa que ha albergado el proyecto al que refiere esta memoria. Intenta no coincidir con ninguna denominación de empresa real, y si lo hiciese, es mero fruto de casualidad. Todos los relatos que se refieren a ella en esta memoria, no corresponden a hechos en tal hipotética organización, sino a una específica re-denominada para los efectos de esta Memoria.
- **Capa**: una capa es una agrupación lógica de código tales que comparten intereses comunes.
- **Capa de aplicación**: también conocida como capa de lógica de negocio, que a su vez es conocida como Capa Lógica. Por ejemplo, en el alta de un nuevo expediente, una vez el usuario pincha sobre el de “crear”, la capa de aplicación interactúa con la capa de datos y

envía la información necesaria a la capa de presentación. La capa de aplicación gobierna las funcionalidades de la aplicación mediante la ejecución de procesos concretos y detallados. Esta capa actúa como mediador entre las capas de presentación y la capa de datos. En términos simples, su función realizar operaciones en la aplicación.

- **Capa de datos**: los datos se almacenan en esta capa. La capa de aplicación se comunica con esta capa para recuperar datos. Contiene métodos que comunican generalmente con una base de datos y realiza acciones concretas sobre los datos, por ejemplo, inserciones, actualizaciones, etc. En términos simples, su función es implementar y recuperar datos.
- **Capa de presentación**: también conocida como capa de cliente. Es la capa más externa de una aplicación. Es la capa visible por el usuario final cuando se utiliza un software. Utilizando esta capa se puede por ejemplo acceder a páginas web. La función principal de esta capa es la de permitir la comunicación con la capa interior, de aplicación. Esta capa transmite la información de un usuario (por ejemplo, acciones de teclado o clics de ratón) a la capa de aplicación. Por ejemplo, un formulario de alta de un nuevo expediente donde los usuarios han de completar una serie de datos (en cajas de texto, listas de selección, etc) y finalmente presionar un botón para persistir los datos. En términos simples, su función es ofrecer un visionado de la aplicación.
- **Caso jurídico** o, simplificando, caso: situación específica que refiere una atención o servicio desplegado en representación de una persona natural o jurídica y que se caracteriza por el desarrollo de un "expediente" propio que le distingue de otros casos.
- **Cliente**: software que solicita servicios de software. (Nota: no se trata de una persona; consulte usuario final). Un cliente puede ser un servicio que solicita otro servicio o un componente de GUI al que obtiene acceso un usuario final.
- **Componente de aplicación**: un componente de software desarrollado de forma personalizada para alguna función informática específica que proporciona servicios de negocios a los usuarios finales o a otros componentes de aplicaciones.
- **Crosscutting Concerns**: es algún interés o preocupación del software (sincronización, registro, asignación de memoria, preferencias de interfaz de usuario, etc) que está en su mayor parte fuera (y ortogonal) al dominio del problema.
- **DataMapper**: es una librería de mapeo relacional escrita en el lenguaje de programación "Ruby" y que sigue el patrón *ActiveRecord*.
- **Empaquetado**: se refiere al acabado final en términos de completitud, y al ordenamiento final que se aplica a la documentación e información contenida en un expediente a efectos de ser presentada en forma física o digital, ante los organismos pertinentes (juzgados), en concepto de base de defensa o de demanda, según corresponda al caso.
- **Endpoint**: Dispositivo de usuario final como, por ejemplo, un ordenador de sobre mesa, un portátil o un teléfono móvil.
- **Expediente**: un expediente puede ser visto como un fichero físico, un archivador, una estructura de tipo informática sobre la que participan una serie de implicados que pueden ser personas físicas (clientes/afectados) y/o jurídicas (organizaciones). Un expediente:

    - Viene caracterizado siempre por el "Tipo de Caso", o también denominado *Producto*.
    - Acopia datos e información recopilada o producida durante su "ciclo de vida".

- **Gestión de la Configuración**: conjunto de procesos destinados a asegurar la calidad de todo producto obtenido durante cualquiera de las etapas del desarrollo de un sistema de información, a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante de una versión estable de cada elemento para toda persona involucrada en el citado desarrollo.
- **Markdown**: lenguaje de marcado ligero creado por John Gruber que trata de conseguir la máxima legibilidad y facilidad de publicación tanto en su forma de entrada como de salida, inspirándose en muchas convenciones existentes para marcar mensajes de correo electrónico usando texto plano.
- **Minería de procesos**: la minería de procesos es una técnica de administración de procesos que permite analizar los procesos de negocios de acuerdo con un registro de eventos. A través de esta actividad se desea extraer conocimiento desde los registros de evento de los procesos almacenados por los sistemas.
- **Persona física**: ser humano que goza de una identidad concreta y que goza de derechos y obligaciones que le sean aplicables de acuerdo a legislación. Es susceptible de ser representada por abogados, en casos jurídicos o en otras instancias, según lo requiera.
- **Persona jurídica**: organizaciones o grupos de personas, legalmente constituidas como unidad independiente de tales personas, que goza de derechos y obligaciones, y es susceptible de ser representada en casos jurídicos. Ejemplo: bancos, entidades públicas, empresas, asociaciones, ONG's.
- **Producto**: refiere de forma genérica a cualquiera de los servicios que realiza *BDA*. La denominación de cualquiera de estos productos se hace indicando la naturaleza del servicio o "Caso jurídico" del que trate. Por ejemplo, "Preferentes", "Gastos de Hipoteca", "Cláusula Suelo", entre otros.
- **Proyecto**: a través de esta memoria el término Proyecto se utiliza con dos acepciones y ámbitos de aplicación:
    - **Proyecto de fin de máster**: que define el conjunto de los asuntos relacionados y orientados a dar cumplimiento a los aspectos curriculares del Máster en Ingeniería Informática realizados por el autor de esta memoria. Esta acepción se nutre en términos de la concreción de sus asuntos de la experiencia y desarrollos realizados en el ámbito de la segunda acepción del término
    - **Proyecto** (empresarial): refiere al conjunto de objetivos, análisis, estudios y desarrollos formulados y realizados para producir una solución informática que resuelve unas necesidades de modo estructurado, coherente y eficaz.
- **Round-Robin**: es un método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento.

- **Servicio**: una función de software realizada para uno o varios clientes. Esta función podría ser de muy bajo nivel, como la administración de memoria, o de alto nivel, como el servicio de negocios de comprobación de crédito. Un servicio de alto nivel puede estar formado por una familia de servicios individuales. Los servicios pueden ser locales (disponibles para clientes locales) o distribuidos (disponibles para clientes remotos).
- **Servicio de negocios**: un componente de aplicación o un ensamblado de componentes que realizan la lógica de negocio en nombre de varios clientes (informáticos). Un servicio de negocio también puede ser un ensamblado de componentes distribuidos encapsulados como un servicio web o puede ser un servidor independiente.
- **Servidor**: un proceso de software con varios subprocesos (a diferencia de un servidor de hardware) que proporciona un servicio distribuido o un conjunto coherente de servicios para los clientes (informáticos) que acceden al servicio mediante una interfaz externa.
    - **Servidor** (físico): se refiere a los dispositivos físicos (computadoras) que proporcionan funcionalidades para otros programas o dispositivos, denominados "clientes".
    - **Servidor** (lógico/virtual): se refiere a una partición dentro de un servidor físico que habilita varias máquinas virtuales dentro de dicha máquina por medio de varias tecnologías.
- **Servicio web**: un servicio que responde a determinados protocolos de Internet estándares [1] para funciones de accesibilidad, encapsulación de servicios y detección.
- **Sniffer**: también conocido como "analizador de red", "analizador de protocolos" o "sniffer de paquetes"; o en determinados tipos de redes, "sniffer ethernet" o "sniffer wireles". Es un programa o dispositivo electrónico capaz de interceptar y registrar el tráfico que circula por una red por parte de ella.
- **Nivel**: o también denominado "tier", es generalmente un límite físico como, por ejemplo, el ordenador de un usuario final o un servidor. Las capas que están diseñadas para ser ejecutadas en un mismo nivel generalmente mantienen una comunicación fina y granular.
- **Usuario final**: una persona que usa una aplicación, a menudo a través de una interfaz gráfica de usuario como, por ejemplo, un navegador de Internet o una GUI de un dispositivo móvil. El número de usuarios finales simultáneos que admita una aplicación es un factor importante de la arquitectura de la aplicación.

2.2 Acrónimos

- **BDA**: Bufete De Abogados, nombre ficticio mediante el cual el autor de este documento se refiere a la organización en donde tienen lugar todos los hechos relatados.
- **BPM**: la Gestión de Procesos de Negocio (en inglés: Business Process Management) es una metodología corporativa y disciplina de gestión, cuyo objetivo es mejorar el desempeño

[1] https://en.wikipedia.org/wiki/List_of_web_service_protocols

(eficiencia y eficacia) y la optimización de los procesos de negocio de una organización, a través de la gestión de los procesos que se deben diseñar, modelar, organizar, documentar y optimizar de forma continua.

- **CRM**: del inglés Customer Relationship Management, sistemas informáticos de apoyo a la gestión de las relaciones con los clientes, a la venta y al marketing.
- **CRUD**: acrónimo de Crear, Leer, Actualizar y Borrar (del original en inglés: Create, Read, Update and Delete)
- **DRBD**: un sistema de replicación para almacenamiento distribuido para Linux. Es utilizado generalmente en clústeres de alta disponibilidad.
- **EAV**: Entity-Attribute-Value, es un modelo de datos para codificar, de una manera eficiente, entidades en las que el número de atributos que se pueden utilizar para describirlos es potencialmente extenso y disperso.
- **FC**: Fibre Channel, canal de fibra es una tecnología de red de nivel físico utilizada principalmente para redes de almacenamiento.
- **HTML**: HyperText Markup Language, lenguaje de marcado para la elaboración de páginas web.
- **HTTP**: Hypertext Transfer Protocol, es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.
- **IIS**: Internet Information Services, es un servidor web y un conjunto de servicios para el sistema operativo Microsoft Windows.
- **JSON**: acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos.
- **iSCSI**: abreviatura de Internet SCSI, es un estándar que permite el uso del protocolo SCSI sobre redes TCP/IP.
- **KVM**: Kernel-based Virtual Machine (en español, Máquina virtual basada en el núcleo), es una solución para implementar virtualización completa con Linux.
- **LACP**: forma parte de una especificación IEEE (802.3ad), permite agrupar varios puertos físicos para formar un único canal lógico.
- **LVM**: es una implementación de un administrador de volúmenes lógicos para el kernel Linux.
- **LXC**: Linux Containers, es una tecnología de virtualización en el nivel de sistema operativo (SO) para Linux.
- **MVC**: Modelo Vista Controlador, es un patrón de arquitectura de software, que separa tanto los datos y la lógica de negocio de una aplicación como la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

- **NFS**: Network File System, es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuido en un entorno de red de computadoras de área local.
- **OOP**: Object-oriented programming , es un paradigma de programación basado en la idea de "objetos", que pueden contener datos, en forma de campos, generalmente conocidos como "atributos" o "propiedades"; y codificados, en forma de procedimientos, generalmente conocidos como "métodos".
- **ORM**: el mapeo objeto-relacional (más conocido por su nombre en inglés, Object- Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional.
- **PHP**: lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.
- **RHEV**: Red Hat Enterprise Virtualization, sistema de virtualización empresarial desarrollado por Red Hat.
- **SDLC**: Software Development Life Cycle, sinónimo de proceso de desarrollo de software.
- **STP**: del inglés Spanning Tree Protocol, es un protocolo de red de nivel 2 del modelo OSI (capa de enlace de datos). Su función es la de gestionar la presencia de bucles en topologías de red debido a la existencia de enlaces redundantes.
- **TFM**: Trabajo Fin de Máster, este documento.
- **ZFS**: es un sistema de archivos y volúmenes desarrollado por Sun Microsystems para su sistema operativo Solaris.
- **VLAN**: acrónimo de virtual LAN (Red de área local virtual), es un método para crear redes lógicas independientes dentro de una misma red física.
- **W3C**: World Wide Web Consortium (W3C), es un consorcio internacional que genera recomendaciones y estándares que aseguran el crecimiento de la World Wide Web a largo plazo.
- **XML**: siglas en inglés de eXtensible Markup Language, traducido como "Lenguaje de Marcado Extensible" o "Lenguaje de Marcas Extensible", es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por *W3C*.
- **XP**: eXtreme Programming, es una metodología ágil de desarrollo de la ingeniería de software formulada por Kent Beck.

CAPÍTULO 3

Presentación de Objetivos y Alcances

En este capítulo se presentan de forma conjunta los objetivos y alcances, tanto de esta memoria como del trabajo fin de máster asociado al mismo.

3.1 Objetivos de Trabajo Fin de Máster y Esta Memoria

1. Presentar la evidencia del cumplimiento por parte de su autor, de los requisitos curriculares para recibir el Máster en Ingeniería Informática.
2. Aportar testimonio y detalle del cumplimiento de un proyecto de fin de Máster en Ingeniería Informática según los requisitos curriculares que le aplican.
3. Constituir una base objetiva para la evaluación del aprendizaje alcanzado por el autor, en las materias propias del Máster en Ingeniería Informática, y de la puesta en ejercicio de dichos conocimientos, en un proyecto que resuelve problemas y necesidades reales de la sociedad y el mercado.
4. Constituir argumento y testimonio de que el autor ha logrado el nivel profesional exigido para adjudicar el grado de Máster en Ingeniería Informática.
5. Constituir material bibliográfico de consulta para otros estudiantes de Grado o Máster en Ingeniería Informática, y otras especialidades a las cuales sean aplicables los temas desarrollados.

3.2 Alcance de la Memoria

Con relación a esta memoria el alcance se restringe al marco de competencias específicas del Máster en Ingeniería Informática del sistema informático usado como foco de desarrollo del *TFM*.

Se excluyen de forma intencionada todos aquellos elementos asociados al ámbito de la ingeniería de software.

Como alcances se señalan los siguientes:

- Pone en relieve aspectos técnicos que deben aplicarse en la gestión de proyectos informáticos de mediana y alta complejidad:
    - Análisis y detección de deficiencias en términos arquitectónicos. Entendido deficiencia, como aquellos elementos disfuncionales y otros necesarios e inexistentes.
    - La concepción de herramientas informáticas orientadas a soluciones;
    - Definición y especificación de arquitecturas física, lógica, de procesos, de código y de datos.
    - Definición de una metodología de desarrollo.
    - Gestión de requisitos, funcionales y no funcionales.
- Tiene como elemento central de referencia el desarrollo de un proyecto real realizado en atención de necesidades de la empresa *BDA*.
- Se despliega a través de:
    - **Un recuento de orden histórico**, que indica la ruta temporal y evolutiva de la propia organización y de la capacidad de identificar el carácter y amplitud de aspectos específicos de dificultades y necesidades que afectan a su actividad tanto en orden interno como en orden de contexto, que determinaron la formulación y realización del proyecto central que refiere (esta memoria).
        - **La Memoria de Proyecto**, que expone la puesta en práctica de los conocimientos y técnicas recogidos en el Máster en Ingeniería Informática por parte del autor:
            - Identificación de dificultades y necesidades;
            - Planteamiento y evaluación de opciones de solución;
        - Desarrollo de herramientas informáticas pertinentes;
            - Ensayos, ajustes e implantación de las herramientas; y
    - **Apartado final de conclusiones**

CAPÍTULO 4

Marco de Condiciones Proyecto Informático de BDA

En este capítulo se describe superficialmente la actividad comercial y la orientación del negocio de *BDA*; el tipo de organización y su funcionamiento interno, y finalmente la situación y contexto tecnológico en la que se ve envuelta *BDA* que motivan la implantación de un sistema de información acorde sus necesidades.

4.1 BDA - Características, Actividad y Contexto

BDA es un despacho de abogados fundado entre el año 2010 y 2015 especializado en derecho bancario y financiero. *BDA* ha conseguido consolidarse como uno de los despachos más destacados del país. Aprovecharon grandes escándalos financieros para abanderar la defensa de los perjudicados y, gracias a una fuerte inversión en publicidad, se les terminó identificando con ello. Su aplicación de la concepción empresarial al ejercicio de la abogacía ha llegado también al modelo de funcionamiento interno de un bufete. En la actualidad cuentan con más de 60 despachos repartidos por todo el territorio español y con más de 600 personas en su plantilla.

4.1.1 Tipo de Organización

El funcionamiento de *BDA* es completamente tradicional y funcional. Se estructura agrupando en departamentos actividades relacionadas entre sí. Su representación suele ser el organigrama, el cual establece la estructura organizativa, designa las funciones de cada trabajador y establece las relaciones jerárquicas (cadena de mando). Sin embargo, el organigrama no muestra el funcionamiento de la organización, las responsabilidades, los aspectos estratégicos, los flujos de información ni la comunicación interna. Esta estructura tradicional, funcional o piramidal, se centra en las necesidades propias de la organización y no en las del cliente, lo cual lleva a “perder” por el camino una

gran cantidad de recursos en actividades que no aportan valor, incrementando considerablemente la burocracia, lo cual multiplica las tareas a realizar. Esta visión departamentalizada genera, a la larga, problemas tales como:

- El establecimiento de objetivos locales o individuales en ocasiones incoherentes y contradictorios con lo que deberían ser los objetivos globales de la organización.
- La proliferación de actividades departamentales que no aportan valor al cliente ni a la propia organización, generando una injustificada burocratización de la gestión.
- Fallos en el intercambio de información y materiales entre los diferentes departamentos (especificaciones no definidas, actividades no estandarizadas, actividades duplicadas, indefinición de responsabilidades, etc.)
- Falta de implicación y motivación de las personas, por la separación entre "los que piensan" y "los que trabajan" y por un estilo de dirección autoritario en lugar de participativo.

Fuente: *[DomingoRey2012]*

4.1.2 Contexto Tecnológico

En la actualidad, en España los procesos de gestión procesal resultan (bajo mi experiencia profesional) complejos y poco cohesionados. Ejemplo de esto es que cada comunidad autónoma, o incluso algunas provincias, proporcionan una serie de herramientas Software orientadas a los letrados para el intercambio de información de una manera segura, entre los distintos órganos judiciales y entre una gran cantidad de operadores jurídicos, y que en su trabajo diario necesitan intercambiar, generar y revisar documentos de tipo judiciales.

Si bien las herramientas que se indican y disponen en el contexto judicial y de abogacía, a priori resultan efectivas, en la mayoría de los casos, para algunos letrados resultan incómodas cuando, por ejemplo, deben trabajar de forma simultánea con distintas plataformas de este estilo, o peor aún, cuando existen casos relacionados entre sí y son gestionados por distintas plataformas de abogacía. En organizaciones tales como bufetes de abogados con presencia en todo el territorio nacional la situación anterior se hace manifiesta. Así, en este documento se expone la situación de una organización real que debe coordinar, de forma simultánea, al menos tres herramientas como las antes mencionadas. Y que por aplicación del principio de "debido cuidado profesional" con enfoque a la protección de datos y asuntos internos sensibles de la empresa en que se realizó el proyecto informático, me refiero a ella con el nombre ficticio *BDA* a efectos de proteger su identidad. Si bien cada una de las oficinas/sucursales de esta entidad se podrían entender como pequeñas entidades con cierto grado de independencia o libertar a la hora de llevar sus gestiones o toma de decisiones, resulta imprescindible obtener una visión global de la organización en su conjunto.

CAPÍTULO 5

Proyecto Informático BDA

En este capítulo se ofrece una breve descripción del proyecto informático *CRM* de *BDA* sobre el cual versa este documento. Se ofrece, además, con la idea de situar el lector en contexto, una pequeña revisión histórica de los acontecimientos relevantes previos y posteriores a mi incorporación y participación en este proyecto.

5.1 Descripción y Denominación del Proyecto

5.1.1 Descripción del Proyecto

Concepción y desarrollo de una herramienta del tipo CRM, aplicada a la gestión segura de información en expedientes de casos jurídicos, en servicios de abogacía, con utilización de Internet, y servicio distribuidos en distintas autonomías y provincias en España.

5.1.2 Denominación del Proyecto

Customer Relationship Management de Bufete De Abogados, o para abreviar, *CRM* de *BDA*

5.2 Justificación, Objetivos y Alcances

5.2.1 Justificación

Las condiciones de cambio y expansión de la actividad y estructura en *BDA*, establecen una condición deficitaria en la capacidad de las herramientas informáticas de gestión.

Un resumen de los principales factores que determinan la falta de adecuación del sistema a las necesidades y propósito de uso sería:

- Colapso recurrente del sistema derivado de déficit en su arquitectura.
- Falta de capacidad para acoger a un número mayor a 200 usuarios de forma simultánea.
- Las capacidades del sistema disponible se restringen a identificación de clientes e información de orden comercial.
- La gestión de casos jurídicos y la documentación asociada en sus respectivos expedientes, elementos centrales de la actividad, no están recogidos ni administrados por el sistema, y se requiere que lo estén para mejorarla productividad.
- Incapacidad del sistema para interactuar con las plataformas externas a las cuales hay que dirigir informes de expedientes, u obtener información.
- Falta de mecanismos de seguridad que garanticen unos niveles mínimos sobre la confidencialidad, integridad y disponibilidad de la información.

En conclusión: ampliar y mejorar capacidades y la confiabilidad del sistema informático de BDA reviste carácter vital, y una necesidad inmediata, que no es posible sustentar a partir de la arquitectura que cuenta el sistema vigente. Una solución sostenible exige un nuevo desarrollo.

5.2.2 Objetivo General

Desarrollar una plataforma informática dedicada al intercambio seguro de información entre las distintas oficinas de *BDA*, y de *BDA* con partes interesadas externas. Con adecuación a las necesidades de la organización, a la naturaleza de la información, a los cuidados y usos previstos para la información, en términos de variedad, eficiencia, y seguridad.

5.2.3 Objetivos Específicos

- Realizar un análisis de diagnóstico orientado al desarrollo del *CRM* de *BDA*.
- Establecer el marco de condiciones, necesidades y requisitos para el diseño, el desarrollo y la implantación eficaz del *CRM* de *BDA*.
- Revisar a nivel arquitectónico las fortalezas, debilidades y necesidades adicionales aplicables al diseño y desarrollo del *CRM* de *BDA*.
- Generar en el orden de diseño conceptual las características y funcionalidades para el *CRM* de *BDA*.
- Realizar las especificaciones técnicas detalladas a cumplir durante el diseño y desarrollo de los distintos elementos del *CRM* de *BDA*.
- Realizar los ajustes y/o mejoras del *CRM* de *BDA* previas a la implantación para uso general.

5.2.4 Alcances del CRM de BDA

- La colección de datos de clientes.
- Gestión de relación con clientes.
- La configuración de expedientes, incluida la anexión de documentos digitalizados, y otros cien por ciento digitales generados durante la gestión del caso jurídico respectivo.
- Gestión de comunicaciones de telefonía y correo electrónico.
- Empaquetado de expedientes y distribución sobre plataformas de terceros.
- Gestión de procesos contables y financieros.

5.2.5 Plan de Gestión

Las actividades comprendidas en el proyecto *CRM* de *BDA* se definieron con un orden lógico en función de siguientes objetivos:

- Identificar el marco de condiciones, y necesidades vigentes y proyectadas, de referencia para el diseño, el desarrollo e implantación del *CMR* de *BDA*.
- Formular objetivos y requisitos aplicables al *CMR* de *BDA*.
- Prevenir errores o déficit del diseño.
- Definir y cumplir la implantación eficaz del *CRM* de *BDA*

Las actividades detalladas para sobre cada objetivo puede consultarse en el apartado “Plan de Gestión” de la sección de *Apéndices*.

5.3 CRM BDA - Relación Histórica de Acontecimientos

5.3.1 FASE 1: Estabilización del sistema preexistente

- Continuas dificultades con su sistema informático convencen a la dirección de *BDA* de que requiere y no dispone y requiere perfiles profesionales idóneos para gestionar sus necesidades informáticas.
- Hasta el momento previo a mi incorporación, el equipo de desarrollo se componía de sólo dos personas con perfil técnico, para un sistema con ya más de 150 usuarios y 17 oficinas distribuidas en igual número de ciudades en España.
- *BDA* solicita los servicios de una empresa externa, con el encargo de mantener una persona con dedicación exclusiva a su sistema.

- Entonces, soy reclutado por esta pequeña empresa de asesorías, con dedicación exclusiva a *BDA*, con la misión de organizar y dirigir el mantenimiento y la mejora de su sistema informático.
- Por aquél primer entonces, las tareas que se me encomiendan son simples y escuetas. Arreglar, mejorar o ampliar son los verbos que mejor las describen. Mi rutina diaria fundamentalmente se restringe a arreglar constantes fallos del "sistema informático".
- El punto de partida para mi trabajo para *BDA* se resume en el cuadro siguiente:
  - No dispongo de nada que describa la estructura o funciones del sistema actual.
  - No dispongo de especificaciones acerca de herramientas o funcionalidades que se requieren o se desea que existan en el corto plazo.
- Una de las primeras tareas claves fue la toma de contacto con el sistema, con el encargo de evaluarlo y determinar que se podía hacer para mejorarlo.
- La revisión exhaustiva y en solitario del sistema me aportó la siguiente apreciación del "sistema informático" existente:
  - En cuanto a las tecnologías, éstas eran las habituales en un entorno web:
    - HTML5, CSS y JavaScritp para frontend;
    - PHP para la programación de páginas dinámicas, backend;
    - MS-Excel y MySQL para el modelado y persistencia de datos; y por último
    - Un servidor web potenciado por IIS7.
  - Sistema soportado sobre infraestructura propia, en instalaciones de *BDA*, mantenido por personal no cualificado; poseen unas competencias sobre un conjunto reducido de actividades de trabajo relativamente simples, y cuentan con unos conocimientos teóricos y capacidades prácticas muy limitadas.
  - La revisión del código fuente, me convenció de estar en frente de un desarrollo "de carácter artesanal o amateur", realizado por personas con bajo nivel de conocimientos estructurados y dominio, tanto en el uso de tecnologías como de los fundamentos para un diseño de software de carácter profesional.
    - Ausencia de procesos para asegurar la calidad del producto,
    - Falta total de herramientas de versionado, y
    - Graves faltas con relación a buenas prácticas.
    - En resumen:
      - Código de baja calidad que dificultaba su interpretación;
      - Estructuras de control de flujo complejizadas innecesariamente, al punto de hacerlas casi incomprensibles; conocidas en sector informático como "código spaguetti".

- Las primeras acciones de mejora se centraron en depurar, normalizar y ordenar datos dentro de la estructura del sistema vigente.
- Un segundo nivel de mejora consistió en desarrollar una primera aplicación en una base de datos relacional:
    - Separando los datos de la planilla Excel principal del sistema, y de otras tantas existentes y dispersas entre las distintas oficinas y departamentos, en tablas de datos ordenadas según tipologías, y
    - Creando las relaciones entre unos y otros datos, y
    - Los formatos y la codificación de múltiples formularios e informes.

Todo lo anterior para sustituir el uso y replicación de formularios y plantillas en Excel, muchas de ellas desarrolladas de modo independiente por los distintos departamentos e incluso por distintas personas para su uso según su particular modo de entender los procedimientos.

- Una vez se conseguido un producto software explotable, mejorado y confiable, se reasignaron las funciones de otras tres personas en las instalaciones de Madrid para realizar pequeñas tareas de mantenimiento y automatización de pruebas del software. Pequeños desarrollos, de impacto reducido, centrados principalmente en las funciones que ya se disponían, sólo que mucho más estable y confiable.

5.3.2 FASE 2: Desarrollo de una herramienta CRM

La segunda fase del proyecto informático *BDA*, consistió en desarrollar una herramienta del tipo *CRM* adaptada a las necesidades específicas de la organización. Que tuvo como génesis el siguiente marco de acciones y condiciones:

- La detección de unas necesidades vigentes y proyectadas ligadas a la expansión de actividad, el incremento de usuarios previsto, y la exigencia práctica de interacción del sistema con plataformas informáticas de uso obligado en el contexto, y
- La decisión de la dirección de BDA de aceptar una propuesta para desarrollar una herramienta informática de amplio alcance y adaptada a la naturaleza de sus procesos y servicios y a necesidades determinadas.
- Al momento de mi incorporación, *BDA* está en plena expansión de su actividad y de su plantilla. Ampliar capacidades y confiabilidad del sistema informático es esencial y una necesidad inmediata.
- Las capacidades del sistema disponible se restringen a identificación de clientes e información de orden comercial. La gestión de casos jurídicos, de recursos humanos, comunicaciones, y otros importantes procesos no están recogidos en el sistema.
- Se presenta a la dirección de *BDA* la propuesta de desarrollar tales capacidades en el sistema, lo cual se acepta.

En esta fase, el diseño completo de todas las arquitecturas para el nuevo sistema de información, constituyen el tema central de esta Memoria de Fin de máster, y se presentan en los capítulos que continúan.

CAPÍTULO 6

Vistas Arquitectónicas

Las vistas arquitectónicas representan un aspecto parcial de una arquitectura de software que muestran propiedades específicas.

En este capítulo de se ofrece una descripción del diseño de alto nivel del sistema *CRM* de *BDA*. Este diseño se presenta por medio de distintas "vistas", cuyo objetivo es mostrar un perspectiva o visión concreta de los distintos elementos que conforman el sistema y cómo éstos interactúan entre ellos. Estas vistas conforman, en definitiva, una proyección de la organización y de sus sistemas informáticos enfocándose en aspectos particulares.

Por motivos de claridad y legibilidad es necesario mencionar que, las secciones siguientes se exponen siguiendo una estructura de tipo "Antes y Después"; exponiendo primero la situación anterior, sus debilidades y, a continuación, la situación actual (momento de la redacción de esta memoria). Detalles como la toma de decisiones, evaluación de soluciones alternativas o cuestiones técnicas que precisen de aclaraciones en mayor profundidad, todas se hayan en forma de documentos independientes en la sección *Anexos* o bien como material transversal en la sección de *Apéndices*.

A pesar de la independencia de todo este material complementario, el autor de éstos es (exceptuando donde se indique) el mismo que el de esta memoria. Algunos han sido creados con el propósito explícito de cubrir las necesidades de esta memoria, otros muchos, producto de las exigencias y/o necesidades laborales dentro en la organización, *BDA*.

6.1 Arquitectura de Código

En una primera etapa el equipo de desarrollo se compone en la práctica de únicamente dos personas, ambos localizados en las instalaciones de Navarra. Si bien existía un flujo de trabajo claro y definido para coordinar a estas dos personas; consistente simplemente en una serie acuerdos verbales sobre cómo proceder en las labores de desarrollo y de gestión de los repositorios de código fuente. La incorporación posterior de nuevos desarrolladores, algunos incluso deslocalizados geográficamente, exigen una definición formalizada y por escrito de este flujo. A continuación, se ofrece un extracto del documento interno que describe, a modo de escuetos procedimientos, la forma de trabajar que ha de seguir cada uno de los miembros del equipo de desarrollo.

Se da por sobreentendido que, los contextos de los elementos expuestos a continuación ocurren bajo un entorno de desarrollo basado en *git* como herramienta de control de versiones. Palabras como "rama", "bifurcar" o "pull request", propias de este contexto, se dan por entendidas y no se ofrecerán mayores detalles al respecto. Ya sea en relación al funcionamiento de esta herramienta de versionado o de sus particularidades, ambas se escapan al alcance de este documento. Para mayor información al respecto se sugiere al lector la lectura de What is Git (https://www.atlassian.com/git/tutorials/what-is-git).

6.1.1 Arquitectura de Código Previa

En sus inicios, el proyecto *CRM* de *BDA*, simplemente no contaba con una arquitectura de código formalmente definida. No existen normas de trabajo que permitan la colaboración, ni se dispone de herramientas consideradas imprescindibles como, por ejemplo, un gestor de versiones que permita una Gestión de la Configuración de forma apropiada. En definitiva, una situación y una gestión del código fuente completamente artesanal e inapropiada.

6.1.2 Arquitectura de Código Actual

Las deficiencias de la organización anterior resultan completamente inapropiadas. No se permite ni facilita la colaboración entre varias personas, no existen trazas de las modificaciones en el código fuente ni planes de despliegue de nuevas versiones.

6.1.2.1 Organización de Ramas

El repositorio está organizado en dos grandes ramas principales:

- *master*: También denominada *producción*, y que representa en último código estable preparado para su despliegue y puesta en marcha en un entorno de producción.
- *dev*: También denominada *desarrollo*. Utilizada por el equipo de desarrollo como base para el diseño e implementación de nuevas funcionalidades, corrección de errores, etc.

En Figura 6.1 refleja de forma generalizada la disposición de ramas y cómo operar con ellas.

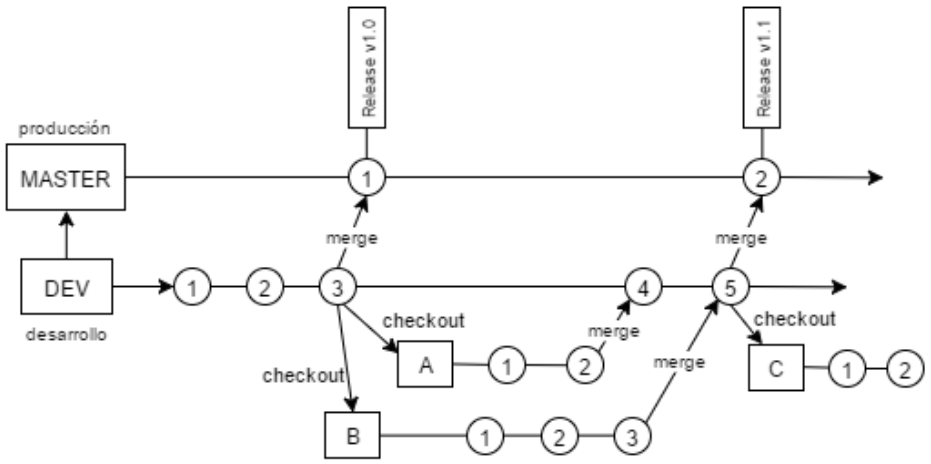


Figura 6.1: Organización de ramas, repositorio de código fuente.
Esquematización del flujo de trabajo básico en el manejo de las ramas de código fuente.

6.1.2.2 Configuraciones Previas

Todo colaborador del proyecto ha de utilizar *git* como sistema de control de versiones. Además, es requisito indispensable configurar la copia local de cada persona de forma que se garanticen unos mínimos de calidad. En concreto, es obligatorio: instalar los *hooks* de *git* para este proyecto, éstos se encuentran en el directorio *ROOT/contrib/* del repositorio, y deben ser copiados en el directorio *ROOT/.git/hooks/* de su copia local del proyecto.

6.1.2.3 Flujo de Trabajo

A continuación, se describe el principio de funcionamiento del flujo de trabajo adoptado en un entorno de desarrollo:

- Primero se debe aclarar que las ramas de producción y desarrollo son consideradas líneas bases, y por tanto **modificaciones directas** de dichas ramas **están totalmente prohibidas**. La modificación de las mismas se debe realizar únicamente mediante la combinación (*merge* y *pull request*) desde ramas bifurcadas/secundarias, y siempre por personal autorizado. Si desea realizar un cambio a estas líneas base, por más pequeño que sea dicho cambio, debe siempre crear una rama nueva como se describe a continuación.
- Cada nueva funcionalidad a implementar, o bien bugfixes a aplicar deben llevarse a cabo en ramas paralelas **bifurcadas desde la rama dev**; puede crear tantas ramas como sean necesarias y bajo el criterio que se estime conveniente, pero siempre utilizando como rama base la rama *dev*. Se sugiere crear una rama por cada nueva funcionalidad a implementar.
- **Debe utilizar siempre nombres descriptivos** para las ramas que bifurque, de forma que el resto del equipo sepa qué se está haciendo. Por ejemplo “refactor-class- XXX”, o “eav- -plugin-fix-YYY”.

- Se debe evitar la creación de *commits* agrupando modificaciones que no guarden ninguna relación. Por ejemplo, si ha refactorizado una clase en un fichero y luego en otro fichero ha simplemente corregido una errata en el *docblock* de un método, **NO debe emitir** un único commit agrupando ambas, pues se tratan evidentemente de tareas bien diferencias.
- Una vez finalizadas las labores de desarrollo se **debe emitir una solicitud de combinación** (*pull request*) a la rama *dev*, dicha solicitud debe pasar todas las pruebas de integración y de verificación antes de ser combinados a la línea base.

En resumen:

- Crear una nueva rama desde *dev* según criterio.
- Trabajar sobre dicha(s) rama(s) (*git checkout*) y realizar todos los *commits* necesarios para la labor de desarrollo.
- Enviar una solicitud de combinación *pull request* hacia *dev*.
- Una vez combinada, eliminar la rama utilizada para el desarrollo.

NOTA: Es posible que durante el desarrollo se encuentre con que su rama de trabajo necesite incorporar *commits* de la rama *dev*, consulte en la base de conocimientos la entrada *Git Rebase* para solucionar la situación.

6.1.2.4 Third-Parties

Como medida de seguridad y aseguramiento de la calidad, aquellas personas ajenas al grupo de desarrollo principal tienen prohibida la modificación (escritura) directa sobre cualquiera de las ramas de este repositorio, poseyendo en su lugar únicamente permisos de lectura sobre el mismo. Por tanto, cualquier sugerencia o incidencia debe ser comunicada a través del **Sistema de Incidencias**, su solicitud será analizada y posteriormente solventada.

Por otra parte, cabe señalar que se admite la **modificación indirecta** del código fuente apoyándose el concepto de *pull request* y cuyas mecánicas operativas quedan descritas a continuación:

- Clonar el repositorio original (*fork*).
- Al clonar, se creará una copia idéntica (al momento de la clonación) del repositorio original en su cuenta personal.
- A continuación, **deberá trabajar sobre su copia personal** del repositorio, y realizar allí todas las modificaciones al código fuente que estime oportunas **siguiendo la misma política descrita en "Flujo de Trabajo"**.
- Una vez finalizadas las labores de desarrollo, deberá enviar una solicitud de combinación de su copia local con el repositorio original, esto se consigue creando un *pull request*.
- Por último, su solicitud será revisada manualmente, y en caso de cumplir con todas las políticas de verificación, será combinada al repositorio original. En caso contrario, se le comunicarán todas las modificaciones necesarias para lograr una combinación exitosa.

NOTA: Cualquier solicitud de combinación, *pull request*, que no se ajuste a las políticas de trabajo definidas en "Flujo de Trabajo" será cancelada e ignorada.

6.2 Arquitectura de Requisitos

En un proyecto software, la importancia de un conjunto de requisitos bien entendidos, priorizados y acordados es imprescindible. Sin embargo, intentar crear un conjunto completo y detallado de requisitos en etapas tempranas de un proyecto resulta contraproducente, restrictivo y excesivo. Resulta simplemente imposible, en proyectos complejos y largo, obtener una visión general y bien definida que permita realizar una definición detallada de los requisitos. El entorno empresarial cambia a medida que el tiempo transcurre; nuevos requisitos y oportunidades aparecen espontáneamente. A medida que el proyecto progresa, el equipo entiende más y más sobre la lógica de negocio y de sus necesidades. Realizar especificaciones detalladas en una etapa temprana significa que, bien habrá que modificarlos después, o bien ceñirse de forma estricta a los requisitos iniciales y por consiguiente no cumplir adecuadamente a las necesidades de negocio.

La especificación es el resultado del proceso de planificación y puede ser vista como un proceso de representación. Generalmente, la especificación del producto describe la visión externa del producto, es una descripción detallada y precisa de la funcionalidad del sistema teniendo en cuenta las restricciones del mismo. Generalmente, la especificación de requisitos sirve como base para el contrato entre los desarrolladores y el cliente. *[AlejandroBedini2013]*

Antes de continuar, se recomienda al lector visitar la sección de *Apéndices*; secciones *Agile Aplicado a BDA*, *eXtreme Programming* y Requisitos una Visión Agile.

6.2.1 Arquitectura de Requisitos Previa

En cuanto a la gestión y toma de requisitos, en un inicio sencillamente no se contaba con una metodología o proceso bien definido. Su ejecución en la práctica no resulta más compleja que una serie de conversaciones orales (con el *stakeholder* responsable del sistema) y un par anotaciones en un soporte físico.

Las deficiencias de este proceder son múltiples y tienen un impacto negativo en el proyecto; continuos retrasos y discusiones por falta de trazabilidad están a la orden del día. Estos requisitos resultan una calidad muy pobre y no exhiben las propiedades de aceptación que caracterizan a aquellos producto de un proceso de ingeniería de requisitos.

Con “calidad” se refiere a especificaciones que manifiestan al menos alguna de estas propiedades: ambiguas, contradictorias, no cohesivas, incompletas, inconsistentes, desactualizadas, especificadas en jerga técnica en lugar de terminologías del usuario o dominio del negocio, imposibles de implementar, superfluas (de baja importancia), irrelevantes para el sistema, falta de meta-datos como prioridades o situación, falta de trazabilidad, no verificables ni validables.

6.2.2 Arquitectura de Requisitos Actual

Vistas las deficiencias en cuanto a la gestión de estos requisitos, se decide implantar *GitHub* como herramienta de apoyo a la gestión de requisitos. Esta herramienta permite solventar alguna defi-

ciencias como, por ejemplo, la falta de meta-datos, situación o trazabilidad. Sin embargo, gran parte de las deficiencias mencionadas en la sección anterior son producto aspectos transversales y fuera del alance de este capítulo como, por ejemplo, el personal encargado de la gestión de requisitos no cuenta con la formación adecuada en el área, asignación inadecuada de recursos o de autoridades para una correcta ejecución de ingeniería de requisitos, acceso inadecuado/limitado a *stakeholders*, entre otros.

6.2.2.1 GitHub como Apoyo a la Gestión de Requisitos

Los requisitos, funciones o no funcionales, son especificados como breves historias de usuario en la plataforma *GitHub*. A continuación, se ofrece una breve descripción del servicio. Como se verá, esta herramienta es el centro neurálgico en lo que al código fuente del aplicativo respecta, y en definitiva en la propia gestión del proyecto en sí, pues el alcance de este servicio resulta mucho más amplio que el puramente ligado al ámbito de la programación. *GitHub* es un servicio en la nube para compartir y publicación código, la "red social de los programadores" como se publicita a sí mismo el servicio. Aunque ambas declaraciones son ciertas, *GitHub* permite realizar mucho más que sólo gestionar código fuente.

En el corazón de *GitHub* se halla *Git*, un proyecto de código libre iniciado por el autor de *Linux*, Linus Trovalds. *Git*, como muchos otros sistemas de control de versiones, gestiona y almacena revisiones de proyectos. Sin embargo, aunque se utilizado principalmente para código fuente, *Git* puede ser empleado una gran variedad de tipos de ficheros, como por ejemplo hojas *Excel*, documentos *Words*, etc. Es decir, *Git* puede ser visto como un sistema de fichero que permite gestionar borradores de ficheros. *[KlintFinley2012]*

GitHub es un servicio de hosting de proyectos, pero ofrece un amplio abanico de funcionalidades. Mientras de *Git* es simplemente una herramienta, *GitHub* por otro lado ofrece una interfaz gráfica web completa. Ofrece además multitud de funcionalidades que agilizan la colaboración entre distintas personas o equipos. Por ejemplo, *GitHub* permite permite gestionar documentación del proyecto de una manera ágil y simplificada gracias a la funcionalidad de *Wikis*, cada proyecto cuenta con su propia Wiki. En el caso de *BDA*, esta funcionalidad se utiliza como base de conocimiento para cada proyecto, ofrece a los miembros del equipo información de interés y relevante dentro de un contexto de desarrollo de software; procedimientos, documentación de sistemas externos, instrucciones de despliegue, etc. Todo esto se puede encontrar en la Wiki de cada proyecto.

GitHub ofrece un sistema de control de incidencias basado en "tickets", denominados *issues*. Una *issue* puede ser entendido como un mensaje que inicia un hilo de conversación, y que consta con una serie de características o propiedades configurables. Para el caso de las historias de usuario, en el proyecto *CRM* de *BDA*, se hace uso de principalmente cuatro de ellas:

- Un *issue*, puede ser clasificado haciendo uso de etiquetas. El conjunto de etiquetas disponibles es configurable y se identifican por un nombre y un color.
- Es posible asignar a uno o varios responsables a un mismo *issue*.
- Un *issue* puede definir un listado de tareas o actividades que han de ser completadas para considerar la "incidencia" como resulta. Evidentemente el uso, o interpretación, que se da a

Figura 6.2: Índice de la Wiki del proyecto *CRM* de *BDA*

Solo personas autorizadas pueden modificar entradas de la Wiki de cada proyecto. El objetivo de esta Wiki es ofrecer, al equipo de desarrollo, indicaciones de despliegue, documentación técnica y procedimientos relevantes necesarios para las labores de desarrollo.

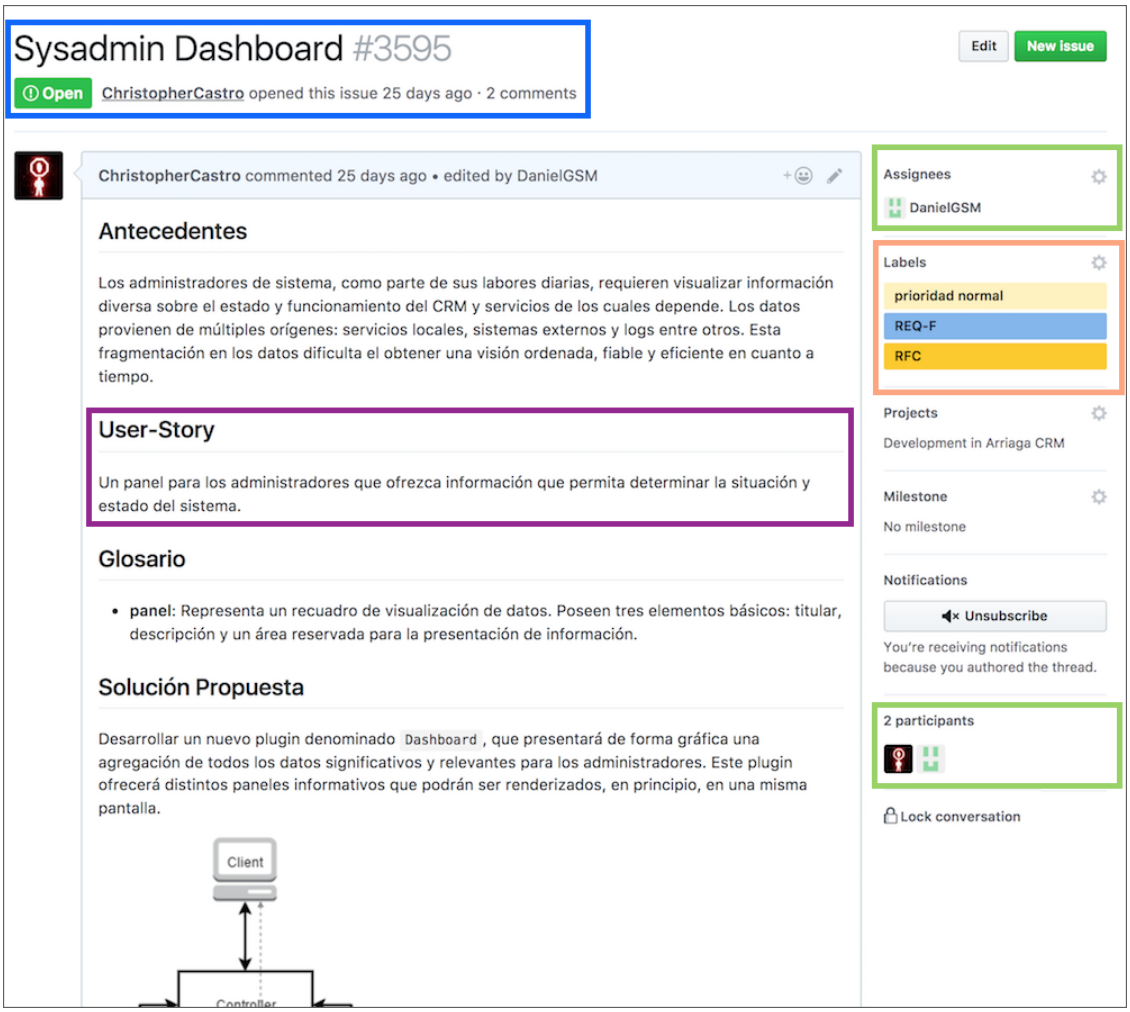


Figura 6.3: Historia de usuario real (púrpura).
Se pueden apreciar los participantes y responsables (verde) además de una serie de etiquetas (naranja) que facilitan la trazabilidad de la historia de usuario. Cada historia de usuario queda identifica de forma inequívoca por un identificador numérico y un titular (azul). Los debates se desarrollan como comentarios (no visibles en la figura) en la propia historia de usuario, a modo de hilo de conversación. Este ejemplo incluye además indicaciones para los desarrolladores.

funcionalidad es la de llevar un control del progreso del requisito.

- Cada bloque de modificación del código fuente (*commit*), puede referenciar a un *issue*. Lo que permite generar una trazabilidad.

De este modo, cada historia de usuario queda representada como un ticket dentro del sistema de *issues*, cada uno de estos tickets refleja tanto la historia de usuario como especificaciones técnicas de desarrollo en caso de ser necesarias. También es necesario señalar que es posible predefinir plantillas, de este modo se marca una línea homogénea y normalizada dentro de la organización en cuanto a la toma de requisitos (y gestión de incidencias); consúltese el apartado de *Anexos* para visualizar las plantillas utilizadas internamente en *BDA*, tanto para la toma de requisitos (historias de usuario), como para el registro de incidencias.

Para cada historia de usuario se indica (haciendo uso de etiquetas) una serie de atributos tales como importancia, estado, tipo de requisito, etc. Estos atributos permiten realizar un seguimiento de cada historia de usuario. Los cambios en los requisitos por parte de los interesados son gestionados mediante "solicitudes de cambios" o *pull-request*, éstos en la práctica no son más que comentarios adicionales al hilo de conversación que define cada historia de usuario. Cada solicitud de cambio es analizada por el gestor del proyecto, quien se encarga de distribuir y planificar de modo que se asegure la integridad del sistema y la correcta adecuación a los estándares de calidad.

Por otro lado, *GitHub* permite integrar sistemas externos a su plataforma mediante una *API Rest*. Así, es posible integrar de forma simple y homogénea mecanismos de verificación y validación automáticas. En el caso de *BDA*, se cuenta con un servidor propio y dedicado a realizar pruebas de integración continua en combinación con la plataforma *GitHub*. De este modo, cada vez que un desarrollo concluye este ha de pasar una serie de pruebas automatizadas para asegurar la calidad y el buen funcionamiento de todo el sistema. Esta característica se puede integrar con otra denominada "Status Check", que permite bloquear la incorporación de cualquier desarrollo que no pase de forma exitosa una serie comprobaciones automatizadas.

6.2.2.2 Project's Boards y Kanban Boards

Por otro lado, en sus ultimas revisiones, *GitHub* ha introducido una nueva funcionalidad denominada *Project's Board*. Estos "Paneles" pueden ser utilizados para definir flujos de trabajo completamente personalizados, como por ejemplo priorizar o identificar desarrollos concretos, crear hojas de ruta completas o incluso listados de actividades. El uso que se le ha dado esta nueva funcionalidad, como se aprecia en Figura 6.5, es la de una herramienta tipo *Kanban Board* y que definitiva permite implementar el método *Kanban* en el proyecto, de este modo se tiene una visión general y simplificada de cada uno de los desarrollos que ocurren dentro del proyecto.

6.2.2.2.1 Metodología de Gestión Aplicada

Es importante aclarar que los *Kanban Boards* son diseñados para el contexto en el que son utilizados, pueden variar de forma considerable y mostrar distintos tipos de elementos de trabajo

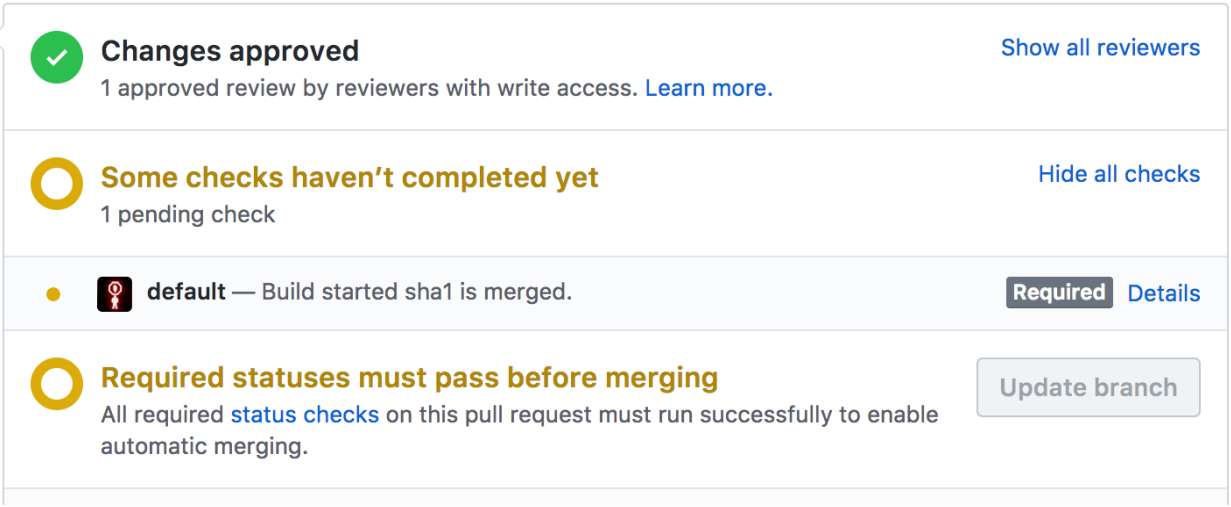


Figura 6.4: Captura de pantalla real de funcionalidad "Status Check" GitHub.
Una solicitud de combinación pull-request no puede ser combinada a no se que el sistema externo Jenkins realice las pruebas de integración pertinentes y las valide.

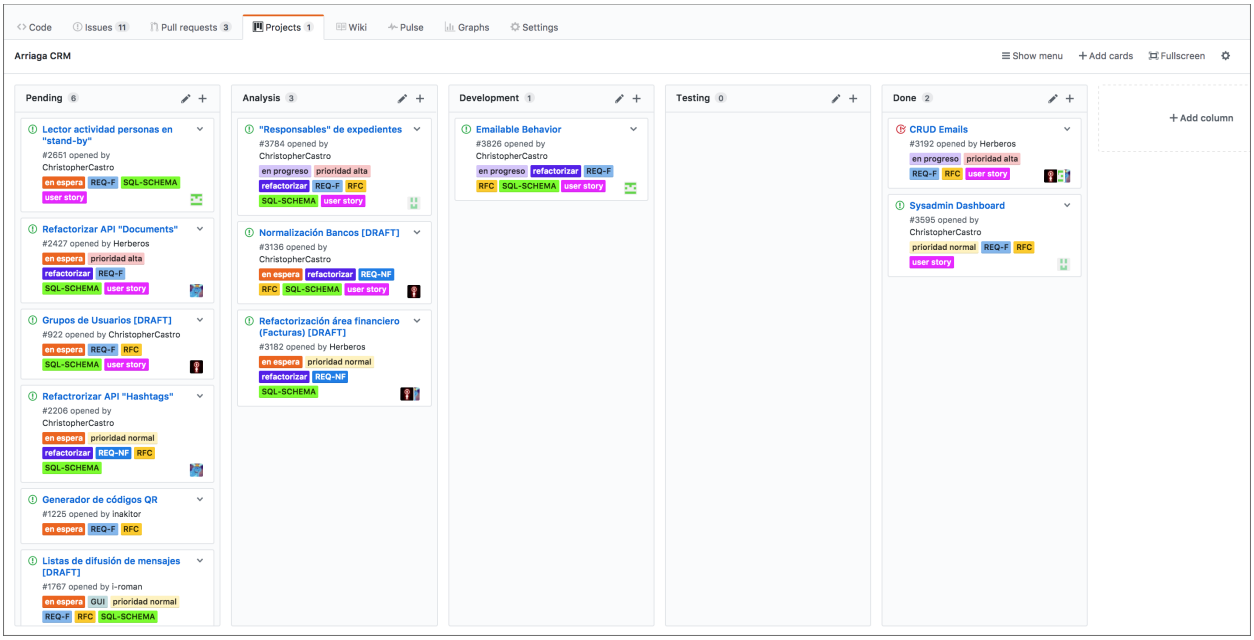


Figura 6.5: Captura de pantalla real de GitHub's Project Board.
Se puede añadir cuantas columnas se deseen, en este caso he decidido utilizar cinco para organizar el flujo de trabajo: "Pending", "Analysis", "Development", "Testing" y "Done".

("características", "user stories", etc). El objetivo en definitiva es que el flujo de trabajo, y el progreso de los elementos individuales, sean claros para los participantes y las partes interesadas. En el caso de *BDA*, se utiliza un *Kanban Board* compuesto por cinco columnas, las actividades evolucionan de izquierda a derecha de la siguiente manera:

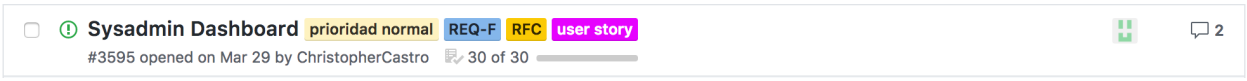


Figura 6.6: Vista resumen de una actividad en concreto.
Se identifica por un título, y se acompañada de una serie etiquetas que permiten su clasificación, un identificador de actividad (#3595), fecha de creación, autor de la actividad y un indicador de progreso de tareas completadas (30 de 30).

- **Pending**: Actividades pendientes de ser analizados con mayor profundidad. Por ejemplo, incidencias técnicas o historias de usuarios.
- **Analysis**: Actividades actualmente en fase de análisis. Aquí es donde se realiza el diseño de soluciones ligadas a historias de usuario. Por ejemplo, en Figura 6.3 se observa una historia de usuario que ha sido "extendida" añadiendo especificaciones de diseño y añadiendo subtareas. En el caso de las incidencias técnicas, aquí es donde se discute la situación y se realizan propuestas para solventarlas.
- **Development**: Una vez el diseño de una solución concluye, la actividad correspondiente se mueve a esta fase. Los desarrolladores pueden utilizar el sistema de comentarios para comunicar el progreso de cada actividad. Pueden también ir marcando sub-tareas como completadas, de este modo es posible realizar un seguimiento del progreso de cada actividad como se observa en Figura 6.6.
- **Testing**: A esta fase se mueven aquellas actividades que han concluido su fase de desarrollo y se encuentran lista para ser probadas en un entorno de producción simulado (sandbox), esto además de las pruebas automatizadas pertinentes.
- **Done**: A esta fase se mueven todas aquellas actividades que han concluido todo el flujo anterior. También se incluyen actividades que puedan haber sido canceladas en cualquiera de las fases anteriores, por ejemplo, desarrollos que se abortan en favor de otros. Como se observa en Figura 6.5, la actividad "CRUD Emails" ha sido abortada y se indica con un ícono de color rojo en la parte superior izquierda.

Artefacto/Característica	The GitHub's Way
Backlog	Listado de incidencias (issues)
User Story	Issue/Ticket/Incidencia etiquetado como user-story
Responsable	Responsable del Issue
Interesados/Participantes	Participantes en el Issue
Trazabilidad	Etiquetas, paneles y referencias en comentarios de commits o pull-requests
Kanban Board	Paneles de proyecto GitHub
Verificación y Validación	Jenkins plugin (sistema externo)
Gráfico Burn up/down	ZenHub plugin (servicio externo)

Tabla 6.1: Equivalencias entre artefactos Agile y GitHub

6.3 Arquitectura Física

En esta sección se ofrece una doble perspectiva de la arquitectura física. Por un lado, se ofrece una descripción simplificada de la arquitectura física inicial, antes de mi participación en este proyecto. Por otro lado, se expone la arquitectura actualmente en uso al momento de la redacción de este documento. De este modo, se ofrecerá una visión de alto nivel de los distintos componentes que conforman el sistema en términos de infraestructura, como, por ejemplo, la disposición física de los distintos equipos hardware.

Se recomienda al lector visitar previamente el apartado de Generalidades, en donde se exponen una serie de conceptos y definiciones claves, de forma que el lector comprenda las motivaciones de las decisiones adoptadas en el proceso de diseño de la arquitectura física.

6.3.1 Arquitectura Física Previa

Las oficinas de *BDA* en Madrid poseen una base de datos donde almacenan la información relativa a los casos jurídicos de todo el territorio nacional, el acceso a esta información se realiza mediante una aplicación desarrollada a medida.

Como puede apreciar en Figura 6.7, y como se mencionó brevemente al inicio de este documento, la arquitectura física inicial al momento de mi incorporación en la organización consistía, en esencia, en un único servidor encargado de prestar los distintos servicios; servicios de correo electrónico, almacenamiento masivo compartido *NFS*, los servicios web relacionados con interfaz de gestión antes mencionada y por supuesto la base de datos entre otros.

Un planteamiento de este estilo, centralizado en una única máquina física, aumenta el riesgo por pérdida de servicio al contar con un punto único de fallo. Sobra decir, Figura 6.8, que las instalaciones físicas no son las más adecuadas ni fiables. Falta de redundancia en el suministro eléctrico o interfaces WAN; una topología de red configurada de forma inapropiada; falta de un sistema de refrigeración adecuado; una evidente falta de organización, de personal cualificado y de planes de

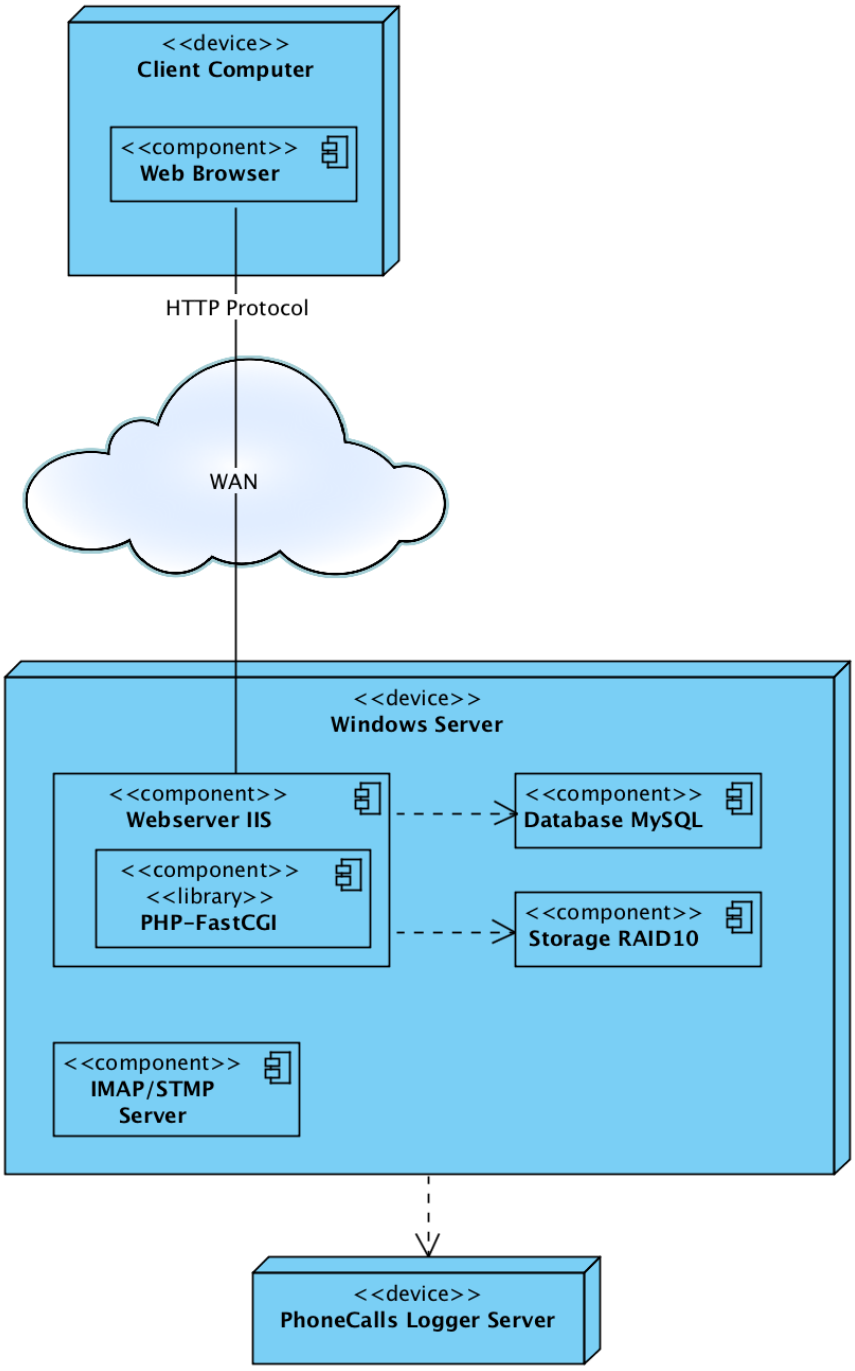


Figura 6.7: Arquitectura física inicial.
Esquematización de la arquitectura física inicial.

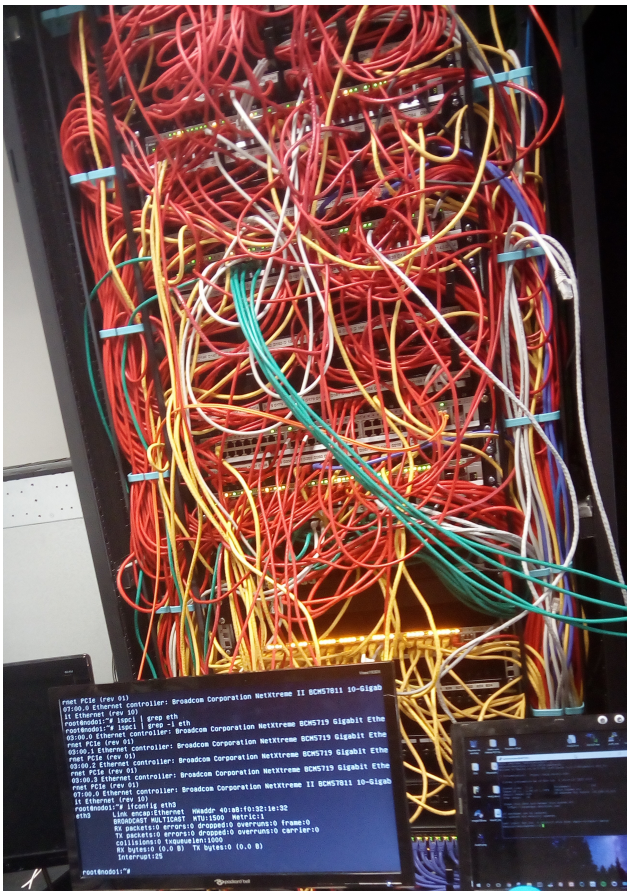


Figura 6.8: Instalaciones oficinas de Madrid, Septiembre 2016.
Uno de los tres Racks en las oficinas de Madrid, ubicado en la planta sexta, alberga servidor de producción y dispositivos de conmutación para *endpoints*. Los otros dos están orientados única y exclusivamente a ofrecer conmutación entre *endpoints* de cada planta e interconectar las mismas.

actuación; todo esto no hacen más que aumentar el riesgo por una posible pérdida de información o incapacidad de prestar servicio.

Una vez vista la situación, se adoptan medidas para, al menos, asegurar la consistencia y disponibilidad de los datos. Si bien cualquier medida que se pudiera adoptar en términos lógicos depende por completo de la situación física que la soporte, al menos reduce el riesgo, aunque no lo elimina por completo.

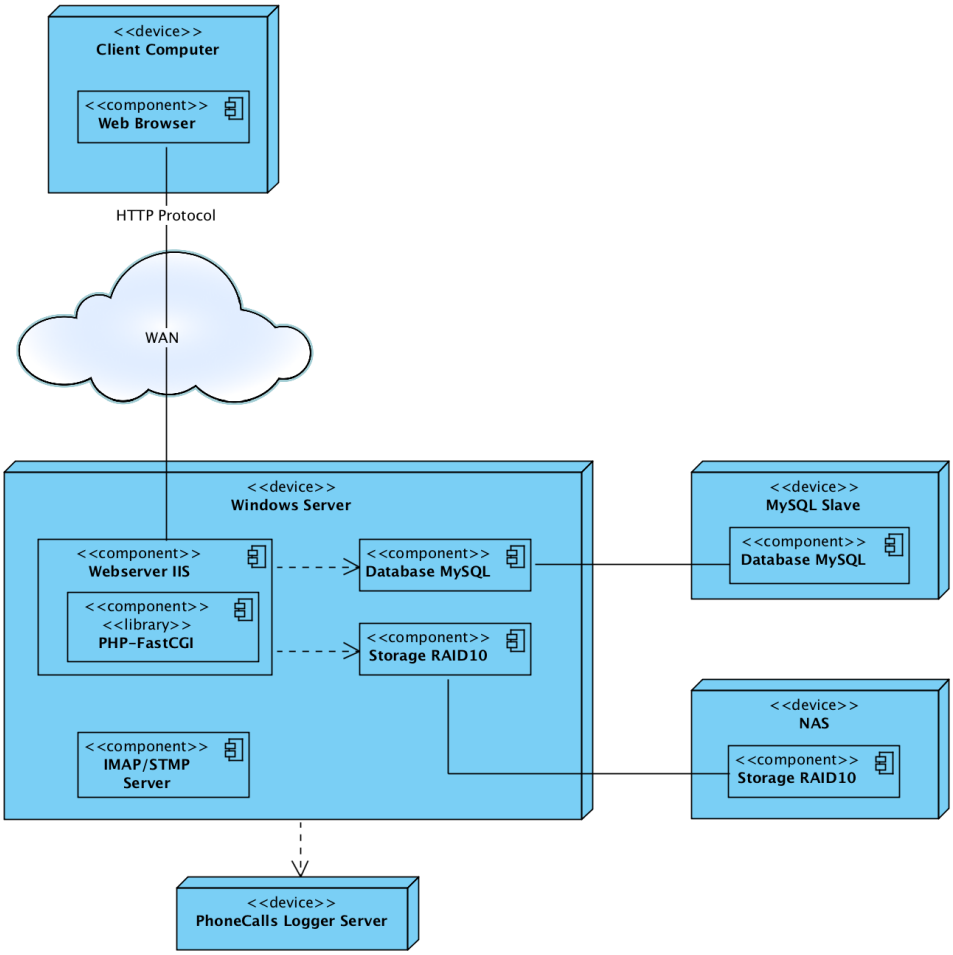


Figura 6.9: Arquitectura física inicial mejorada con replicado.
Replicado de los datos considerados críticos en máquinas aisladas e independientes.

Esta primera medida consiste en realizar un replicado de la información en máquinas físicamente aisladas. En concreto interesa asegurar dos elementos: la base de datos y el almacenamiento masivo. Para el primero se opta por configurar un replicado semi-síncrono de la base de datos MySQL en una segunda máquina física. Para el segundo elemento, se cuenta con la ventaja de que se trata de un almacenamiento incremental, lo que facilita las labores de mantener en un estado consistente las posibles replicas; se opta por configurar un replicado incremental en tiempo real del almacenamiento masivo en una tercera máquina y de este modo reducir el riesgo por fallos catastróficos del soporte físico. Una vez aplicadas estas medidas la arquitectura se asemeja a la que se puede

observar en Figura 6.9. Es importante recalcar que bajo este nuevo escenario no existe ningún mecanismo de conmutación por fallos automático (failover [1]), ante algún fallo catastrófico del equipo principal es necesario detener los servicios.

6.3.1.1 Topología de Red

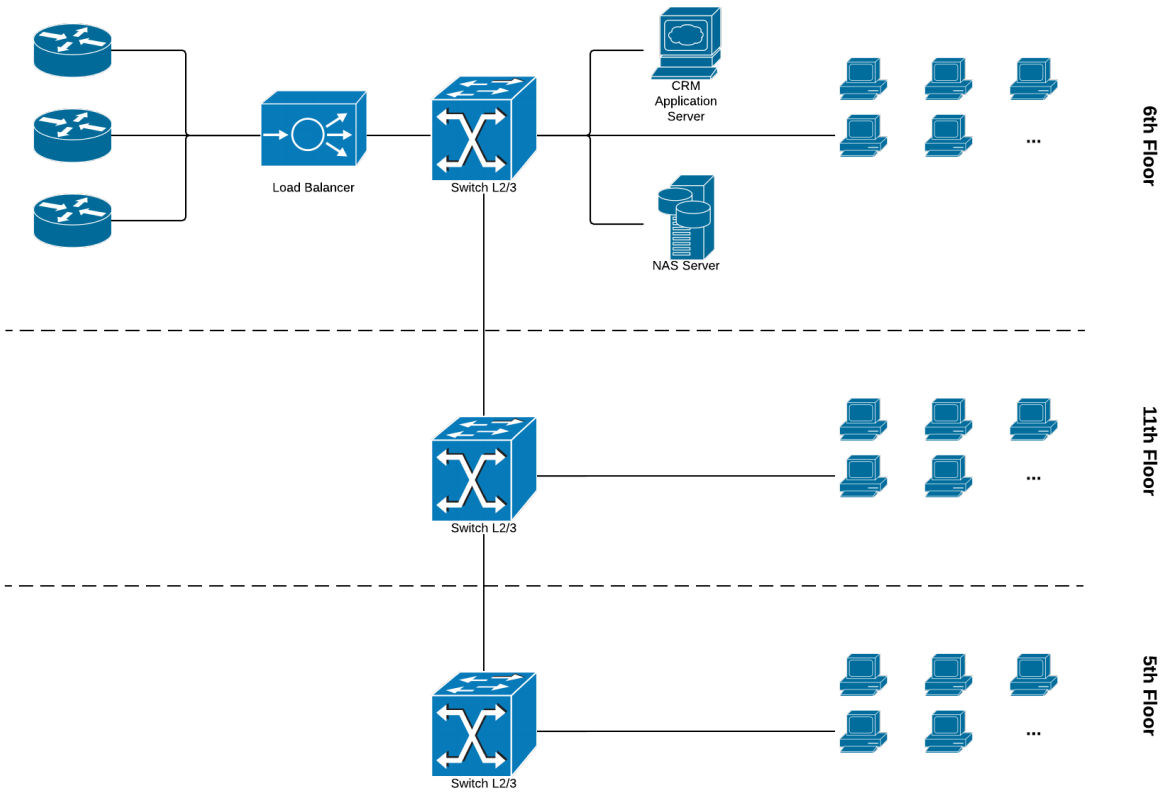


Figura 6.10: Mapa topológico (físico) de red. Instalaciones Madrid, septiembre 2016.
Toda la organización funciona bajo una única sub-red interna, 192.168.32.0/40, no existe ningún tipo de aislamiento ni por plantas o departamentos. Todos los enlaces del diagrama son de tipo 1Gbit. No existe redundancia de enlaces de ningún tipo y toda la red de conmutación está configurada en serie, por ejemplo, un fallo de la línea entre alguna de las plantas produce un aislamiento completo entre ellas.

En cuanto a la topología de red, ésta se mantiene totalmente inalterada a pesar de las medidas adoptadas descritas en la sección anterior; en ambos casos la topología es la observada en Figura 6.10. Como se comenta en la misma figura, esta configuración posee multitud de deficiencias.

[1] La tolerancia a fallos o conmutación por error (en inglés: failover) se refiere a la capacidad de un sistema de seguir funcionando, aún en caso de producirse algún fallo en el sistema

6.3.1.2 Deficiencias

En 6.2 se expone de forma clara y resumida las principales deficiencias y debilidades detectadas en el planteamiento inicial, tanto en la disposición de los servidores como en lo relativo a la red.

6.3.2 Arquitectura Física Actual

La situación inicial resulta completamente insuficiente en términos de escalabilidad y fiabilidad. A pesar de las medidas preventivas adoptadas en Figura 6.9, el riesgo de interrupción del servicio ocasionadas por el fallo de uno de sus elementos principales sigue estando presente. Para hacer frente a esta problemática se decide migrar hacia un entorno virtualizado de nube privada; las máquinas encargadas de prestar los servicios principales son completamente etéreas.

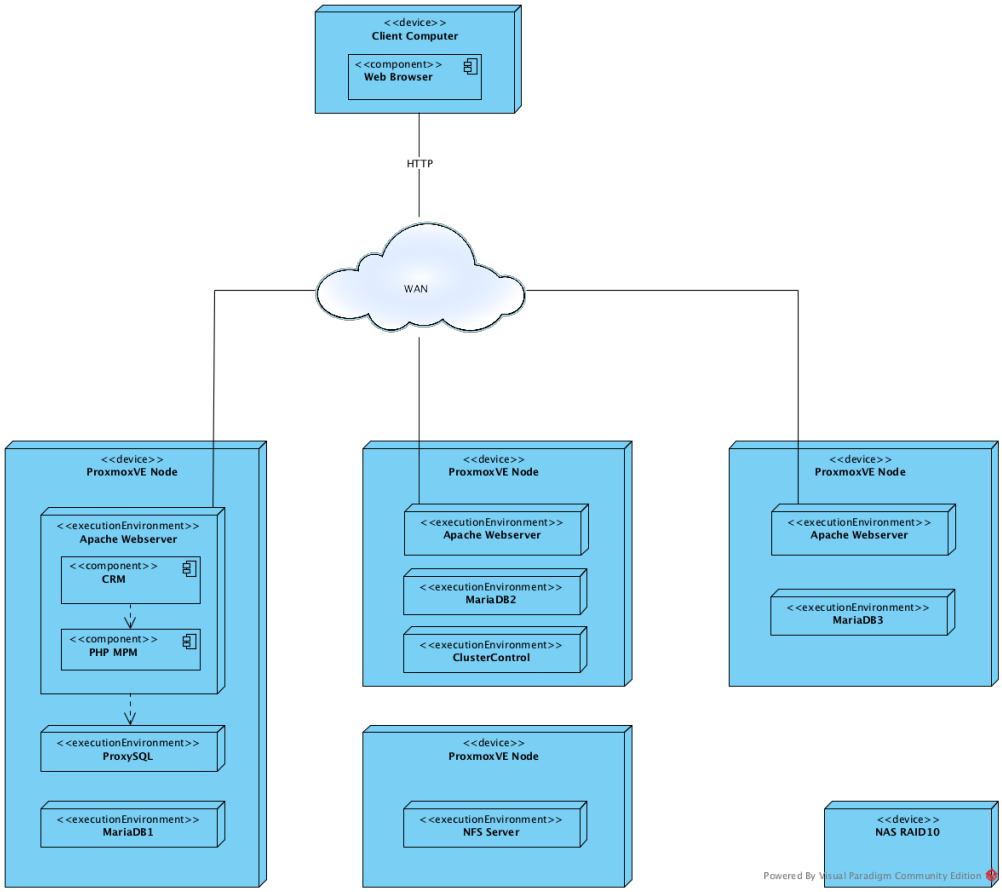


Figura 6.11: Arquitectura física actual. Pamplona, año 2017.
Todas las máquinas han sido virtualizadas. Además, se ha incorporado un sistema de alta disponibilidad maestro-maestro para MySQL, MariaDB Cluster. Algunas relaciones entre componentes han sido omitidas para no entorpecer el visionado, consulte la vista lógica para conocer en mayor detalle cómo interactúan entre sí los distintos elementos.

Por otro lado, es importante subrayar que, en relación a la arquitectura física actual, toda la in-

ARQUITECTURA DE RED	
Deficiencia	**Descripción**
Segmentación física entre plantas con un único punto de acceso WAN	La salida al exterior, WAN, sólo es posible a través de la planta sexta.
Plantas interconectadas en serie con enlaces no redundados	Esto combinado con el punto anterior produce, en ocasiones, un aislamiento completo en situaciones donde, por ejemplo, en enlace entre la planta sexta y decimoprimera se interrumpe.
Ausencia de sub-redes	Toda la organización está configurada para funcionar bajo una única sub-red (192.168.32.0/40), no existe ningún tipo de aislamiento en este sentido. Cámaras de seguridad, servidores considerados críticos, etc; todos visibles y accesibles desde cualquier punto de la red de la organización.
Dispositivos configurados de forma inapropiada	A pesar de contar con un equipamiento de conmutación de red de última generación, el uso que se da a estos dispositivos es el que se le daría a cualquier dispositivo doméstico. Por ejemplo, no se cuenta con mecanismos activos para la detección de bucles como STP, lo que produce situaciones inverosímiles, como que una persona dentro de la organización conecte ambos extremos de un cable Ethernet y toda la organización quede fuera de servicio.
ARQUITECTURA DE SERVIDORES	
Deficiencia	**Descripción**
Nodo único	Un planteamiento monolítico, con todos los servicios centralizados en una única maquina física, aumenta el riesgo por pérdida de servicio al contar con un punto único de fallo.
Falta de redundancia	Relacionado con el punto anterior, los elementos considerados críticos no están redundados de ninguna forma.

Tabla 6.2: Deficiencias arquitectura física inicial

fraestructura de servidores ha sido trasladada físicamente a las instalaciones de Pamplona. Las motivaciones de esta decisión quedan fuera de mi alcance y se deben principalmente la falta de personal cualificado requerido para mantener las instalaciones de Madrid. Constantes incidencias e incapacidad para resolverlas, combinado con una falta de control en términos de seguridad, convierten a las instalaciones de Madrid en un entorno poco apto y fiable para el despliegue de las infraestructuras necesarias.

El escenario físico actual, al momento de redactar este documento, es el observado en Figura 6.11. Los criterios de elección del entorno tecnológico se guían por la idea de disminuir el riesgo lo máximo posible y conseguir un sistema altamente disponible. Riesgos entendidos como aquellos que puedan surgir durante el funcionamiento del software, por ejemplo, mal funcionamiento de algún elemento hardware, alguna de las herramientas de desarrollo/monitorización o la no disponibilidad de los servicios. Se opta por una plataforma de virtualización bien conocida, contrastada y gratuita: ProxmoxVE, se invita al lector a visitar la sección de *Apéndices* para información detallada sobre este producto y el proceso de decisión/selección entre distintas alternativas disponibles.

En Figura 6.11 se describe la disposición física del hardware utilizado para desplegar los distintos servicios. De este modo, y considerando que las máquinas que prestan estos servicios quedan ahora virtualizados (y por tanto en cierto modo "ocultos" en términos físicos), Figura 6.12 proporciona una visión con un mayor nivel de abstracción. Se invita al lector a consultar la sección de anexos, *Relación Hardware y Software*, para conocer en mayor detalle las especificaciones de hardware de cada dispositivo.

Los usuarios se conectan a la plataforma utilizando un enrutador HTTP en la nube. Este enrutador redirige a cada usuario a alguno de los tres servidores web (Apache1-3) siguiendo un algoritmo Round-Robin. A su vez cada servidor web interactúa de forma indirecta con la base de datos a través de un enrutador SQL, ProxySQL, redundado de forma pasiva con conmutación automática en caso de error. La base de datos es un cluster de replicación maestro-maestro síncrono basado en MariaDB (una versión modificada de MySQL), es decir, es posible escribir en cualquiera de los nodos que conforman el cluster. Sin embargo, esto puede ocasionar ciertos problemas con las escrituras en un entorno altamente concurrente (como es el caso de *BDA*), así se ha decidido configurar ProxySQL de modo que las escrituras se realicen siempre en una misma instancia y las lecturas sobre cualquiera de ellas, esta configuración se conoce como "read-write splitting".

Por último, el almacenamiento masivo *NFS* queda encapsulado en una máquina virtual (*NFS* Server) cuya única finalidad es exponer un servicio *NFS* a las maquinas ApacheWebServer sobre una red de alta velocidad a 10Gbps. Las modificaciones sobre este almacén se propagan de forma asíncrona a un segundo almacén en la red (NAS RAID10), de forma que se tenga siempre en todo momento una copia lo más actualizada posible en caso de que *NFS* Server deje de prestar servicio.

6.3.2.1 Topología de Red

En cuanto a la topología de red, como se observa en Figura 6.13, como principales elementos destacables están la redundancia de enlaces utilizando el protocolo *LACP*, y por otro lado la segmentación lógica de la red utilizando *VLAN's*. Si bien se continúa con un único punto de acceso

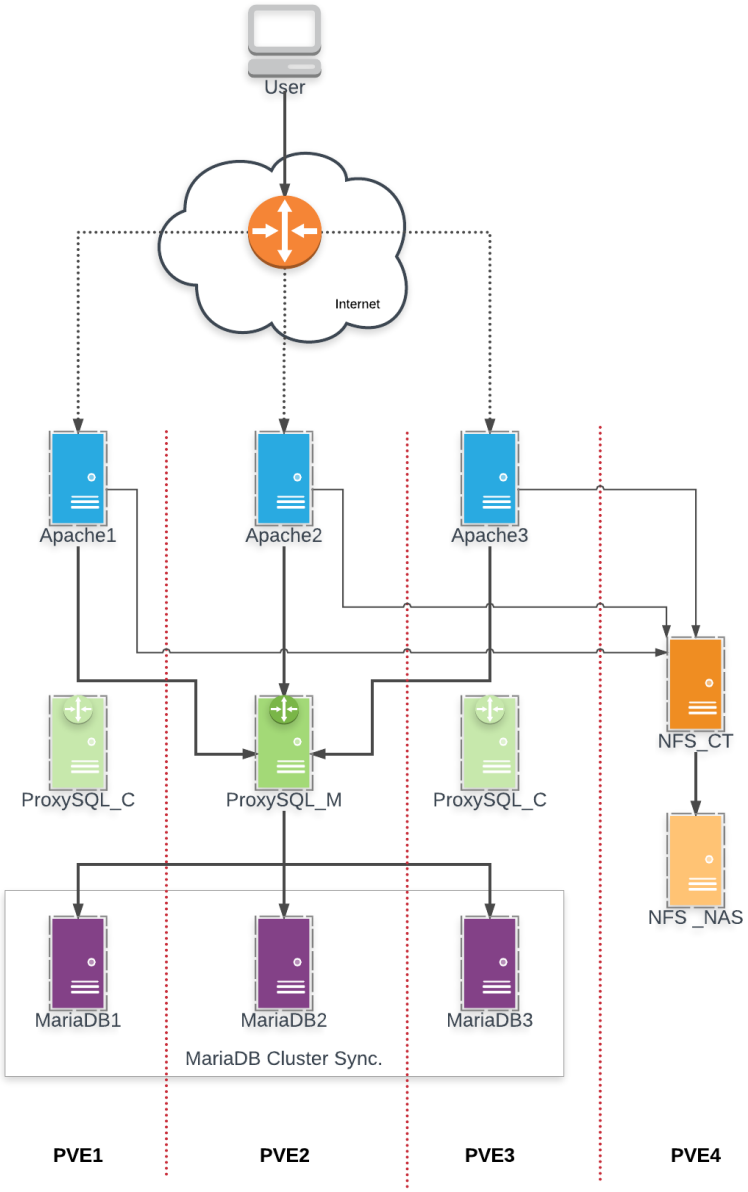


Figura 6.12: Arquitectura física y virtual. Pamplona, junio 2017.
Las separaciones verticales representan divisiones físicas. Por ejemplo, las máquinas Apache1, ProySQL_C y MariaDB1 se encuentran virtualizadas físicamente en una misma máquina anfitrión (PVE1).

WAN (balanceador), la redundancia de los enlaces reduce el riesgo de aislamiento hacia el exterior por fallo de un enlace troncal como ocurría en Figura 6.10.

Por otro lado, es importante destacar la redundancia física del núcleo de conmutación principal (Switch L2/3 Stack), pues se trata en realidad de un conjunto de múltiples equipos que conmutación que se comportan lógicamente como un único switch. Esta tecnología es ofrecida por el fabricante de los equipos (D-link) bajo el nombre de "Virtual Stacking" y permite gestionar todo el conjunto de switches como si fueran una única unidad. En Figura 6.14 se observa la configuración utilizada en *BDA*.

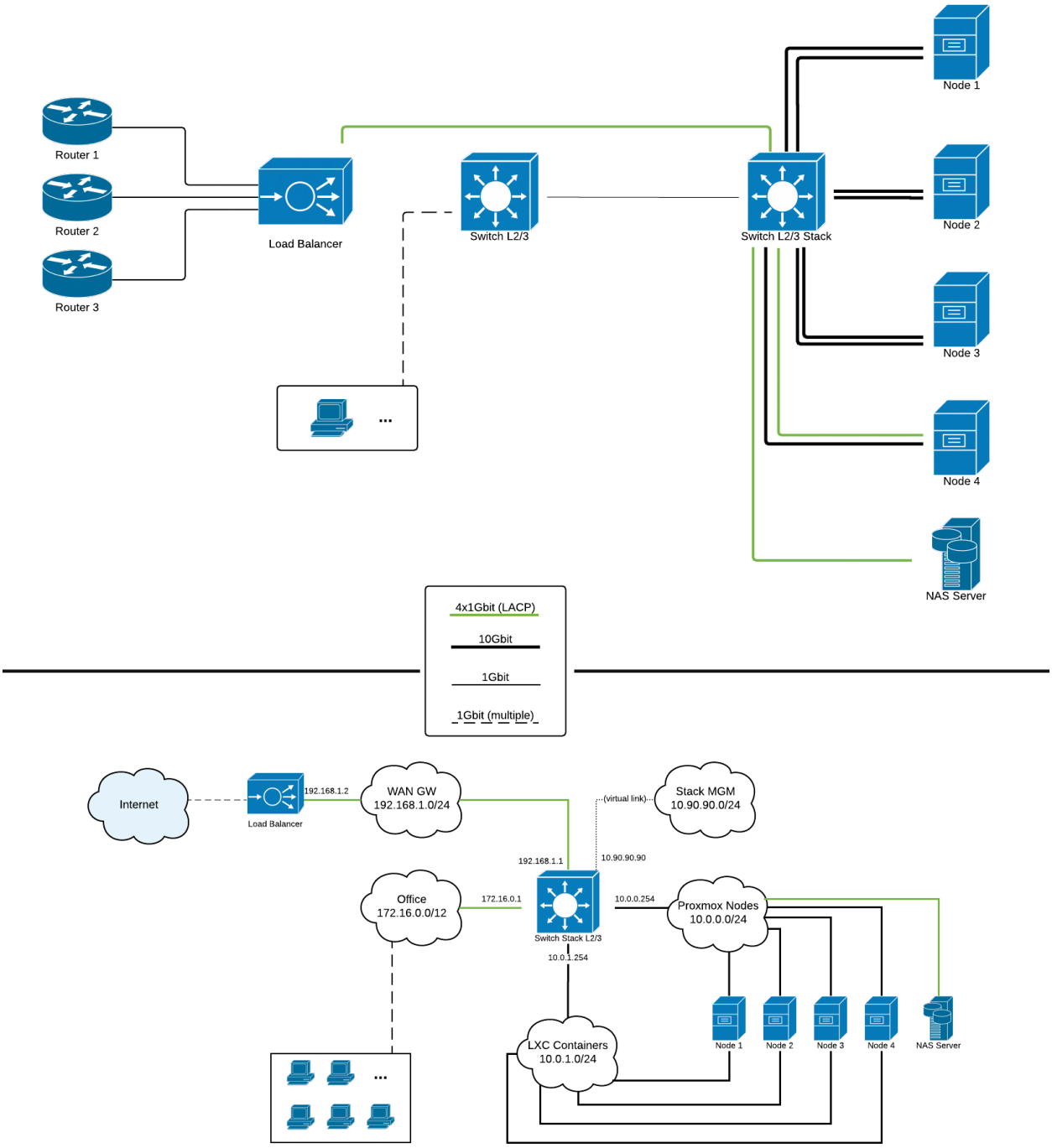


Figura 6.13: Mapa topológico (físico arriba, lógico abajo) de red. Instalaciones Pamplona, Febrero 2017.

Como principales novedades: segmentación lógica de la red empleando VLAN‘s y redundancia de enlaces usando protocolo LACP.

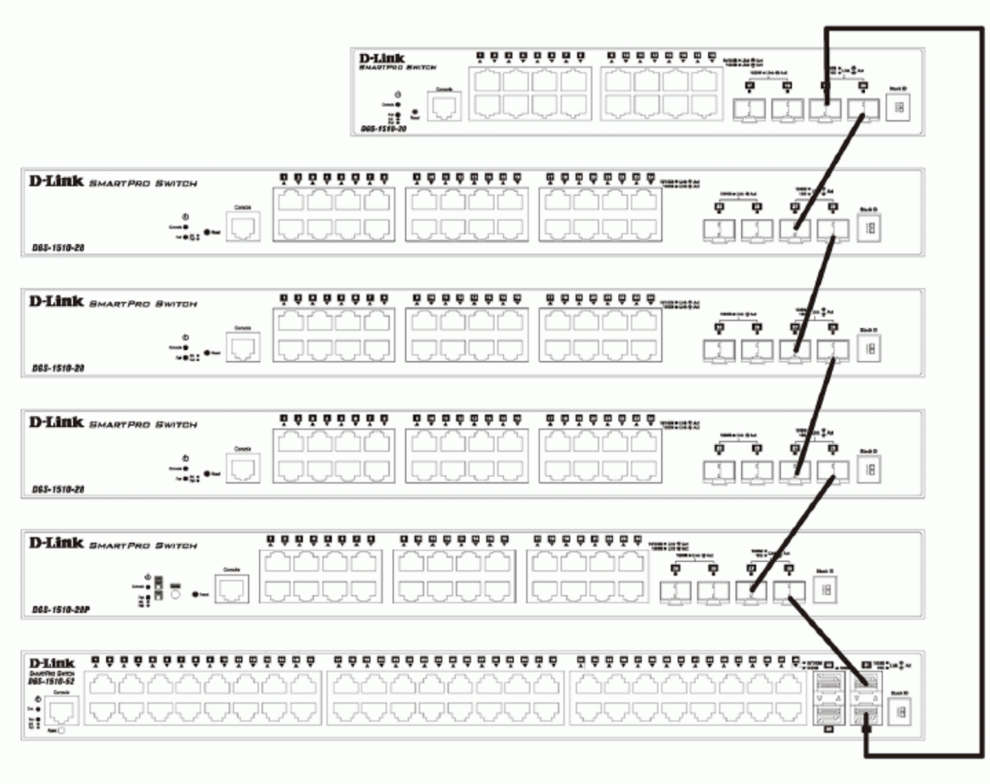


Figura 6.14: Switches apilados en modo “Duplex Ring”.

El modo “Duplex Ring” apila hasta 12 switches en un formato de tipo anillo o circulo, sobre el cual los datos pueden circular en dos direcciones. Este tipo de configuración otorga una gran elasticidad por el hecho de que si parte del anillo deja de funcionar, entonces los datos pueden seguir siendo transmitidos por los cables del anillo cruzando los switches.

6.4 Arquitectura Lógica

En esta sección se enfoca en la descripción interna de los distintos sistemas, es decir, una descripción de sus componentes lógicos que participan en la solución software foco de estudio.

Por un lado, se expondrá brevemente la arquitectura lógica inicial del sistema junto con su problemáticas y deficiencias. A continuación, se expone una descripción de la arquitectura lógica actualmente en uso, la cual intenta solventar los problemas de la anterior. En ambos escenarios la arquitectura sigue, a un alto nivel, un modelo cliente- servidor. La aplicación se divide en un primer componente denominado servidor, encargado de prestar servicio, y un segundo componente denominado cliente, quienes usan los servicios expuesto por el primero. Un cliente viene representado por cualquier sistema externo capaz de seguir el protocolo HTTP, generalmente un navegador web; indistintamente de donde se encuentre éste, un ordenador de sobremesa o un dispositivo portátil, por ejemplo. Por otro lado, el componente denominado servidor es donde se instala el sistema *CRM* de *BDA*, y que ofrece servicios como la búsqueda de expedientes jurídicos; autenticación y autorización de usuarios; búsqueda de clientes/afectados y en definitiva los distintos servicios permiten la gestión integral de los distintos casos jurídicos administrados por la organización.

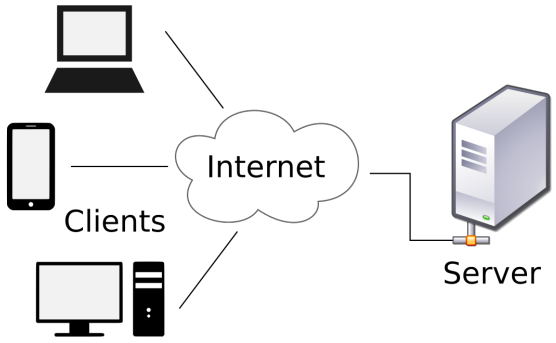


Figura 6.15: Un diagrama cliente-servidor vía Internet.
La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Fuente: Wikipedia.

Una de las ventajas de utilizar este modelo arquitectónico es que se permite la distribución de sus componentes de una manera sencilla, lo que otorga cierta elasticidad al facilitar el escalado y permitir, por ejemplo, que se puedan incorporar nuevos servidores en el hipotético caso de que el número de usuarios de tipo clientes de la aplicación crezca, esto de una manera completamente transparente de cara a la aplicación.

6.4.1 Arquitectura Lógica Previa

La arquitectura inicial sigue una estructura monolítica, en donde todos los componentes se agrupan tanto física como lógicamente en una misma unidad. Es decir, una arquitectura *Tier-1*; una única

capa que mantiene todos los elementos (interfaces, middlewares y backed-end) de la aplicación. Los desarrolladores a menudo ven este tipo de sistemas como simples y más directos. Algunos expertos los describen como aplicaciones que pueden ser instaladas y ejecutadas en una única máquina.

Figura 6.16: Arquitectura lógica inicial.
Todos los componentes se agrupan tanto lógica como física en un único lugar. La separación entre capa de presentación y de negocio es difusa o inexistente. No existen interfaces bien definidas para el acceso a la capa de datos, y ésta última consiste en esencia en ficheros MS Excel.

La necesidad de modelos distribuidos en sistema web y sistema "en la nube" han creado muchas situaciones en donde soluciones arquitectónicas tipo *Tier-1* son simplemente insuficientes. Esto provoca que soluciones *Tier-3* o *Multi-tier* se vuelvan más y más populares. Los beneficios de una arquitectura multi-capa son evidentes: ofrecen una mayor seguridad, mejor rendimiento y una facilidad para escalar. Sin embargo, la adopción de una arquitectura de capa única puede estar motivada por los costes involucrados, en donde puede tener un mayor sentido mantener una aplicación más simple contenida en una única plataforma más sencilla. Se invita al lector a consultar el apartado Generalidades para más información sobre las distintas capas y sus funciones.

6.4.1.1 Deficiencias

En 6.3 se expone de forma clara y resumida las distintas deficiencias y debilidades detectadas en el planteamiento inicial, tanto en la disposición de los servidores como en lo relativo a la red.

6.4.2 Arquitectura Lógica Actual

Como se comentó al inicio de este capítulo, todo el sistema adopta el modelo cliente- servidor como base de su arquitectura. En lo concerniente al lado servidor, a un nivel más bajo de abstracción lógica, la aplicación está desarrollado sobre el framework *MVC CakePHP*. Para mayor información sobre esta herramienta se invita al lector a visitar la sección de *Apéndices*.

Deficiencia	Descripción
Distribución	No permite acceso remoto o distribuido a los recursos.
Mantenimiento	Planteamiento monolítico que imposibilita el mantenimiento del sistema.
Escalabilidad	No permite escalar para prestar servicio a un número mayor de usuarios.

Tabla 6.3: Deficiencias arquitectura lógica inicial

6.4.2.1 Capas de Servicio

Los distintos componentes de software de la nueva arquitectura se pueden visualizar como encapsulados en un determinado número de capas de servicio. Estas capas representan la independencia física y lógica de los componentes basándose en la naturaleza de los servicios que estos proporcionan. Las capas de servicio de la nueva arquitectura se muestran en Figura 6.17. A continuación se proporcionan breves descripciones de las tres capas que se muestran en la figura. Cabe aclarar antes que: una capa de servicio representa una agrupación conceptual de capas lógicas software (en función de los servicios que éstos prestan). A su vez, estas capas lógicas representan agrupaciones de componentes software fuertemente acoplados.

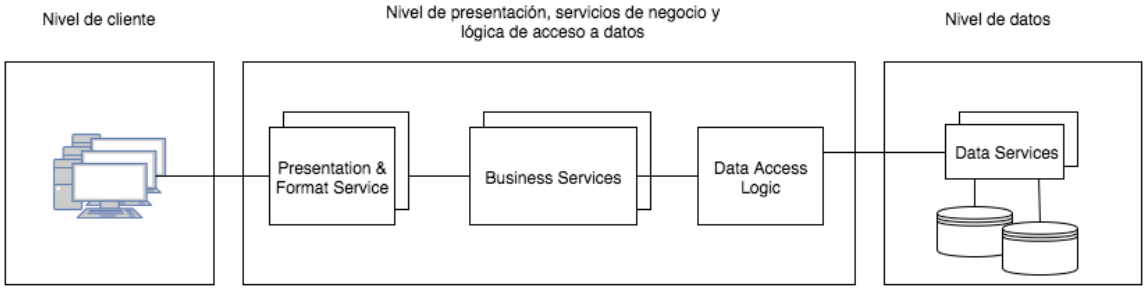


Figura 6.17: Capas de servicio la aplicación *CRM* de *BDA*.
Cada capa contiene diferentes agrupaciones lógicas de la aplicación. Por ejemplo, la capa de presentación representa una agrupación de componentes fuertemente acoplados dedicados a labores de formateo y presentación de información.

- **Capa de cliente**: La capa de cliente encapsula los servicios de la aplicación a la cual tiene acceso directamente un usuario final mediante una interfaz de usuario. Esta capa puede incluir clientes basados en navegadores web, dispositivos móviles o en definitiva cualquier sistema compatible con el protocolo *HTTP*, no necesariamente un humano.
- **Capa de presentación, negocio y acceso a datos**: Esta capa de servicio encapsula tres grandes capas lógicas bien diferenciadas:
    - La capa de presentación, está formada por la lógica de aplicación que prepara los datos para su envío a la capa de servicio superior y procesa solicitudes desde ese mismo nivel

para su envío a la lógica de negocios del servidor. La lógica en la capa de presentación consiste normalmente en componentes que preparan los datos para enviarlos en formato *HTML*, *JSON*, *XML* o que reciben solicitudes para procesarlas.

- La capa de negocio, consiste en aquella lógica de la aplicación que realiza las funciones principales de la misma: procesamiento de datos, implementación de funciones de negocios, coordinación de varios usuarios y administración de recursos externos como, por ejemplo, bases de datos. Está formada por componentes software fuertemente acoplados.
- La capa de acceso a datos, consiste en aquella lógica que permite una comunicación con las capas del nivel inferior. Ofrece mecanismos e interfaces para la recuperación y manipulación de información con los distintos servicios disponibles en el nivel de datos. Por ejemplo, interfaces para trabajar con bases de datos relacionales, para base de datos no relacionales o incluso mecanismos para operar con tecnologías de almacenamiento masivo basado en objetos.

- **Capa de datos**: La capa de servicio de datos consta de servicios que ofrecen datos persistentes utilizados por la lógica de negocio. Los datos pueden ser datos de la aplicación almacenados en un sistema de administración de bases de datos o pueden incluir información de recursos y directorios almacenada en un almacén de datos como, por ejemplo, un servicio *NFS*.

6.4.2.2 Independencia Lógica y Física

En Figura 6.17 se aprecia la independencia lógica y física de los componentes, representada mediante capas de servicio bien definidos y separados. Estas capas de servicio representan la partición de la lógica de la aplicación en varios equipos físicos en un entorno de red y de hardware como el descrito en *Arquitectura Física*:

- **Independencia lógica.** Las tres capas de servicio del modelo arquitectónico representan la independencia lógica entre ellas. De este modo es posible modificar la lógica de la aplicación concerniente a una capa de servicio en concreto de forma independiente al resto de capas. Por ejemplo, es posible modificar la implementación de la lógica de negocios sin necesidad de modificar o actualizar la lógica contenida en la capa de servicios de cliente. Esta independencia significa, por ejemplo, que es posible introducir nuevos tipos de componentes de clientes sin tener que modificar los componentes de los servicios de negocios.
- **Independencia física.** Las tres capas de servicio también manifiestan una independencia física. Es posible desplegar la lógica de cada una de estas capas en distintas plataformas hardware, es decir, distintas configuraciones de procesadores, memoria, sistema operativo, etc. Esta independencia facilita la distribución de sus servicios y componentes sobre equipos hardware especializados, de forma que se adapten a las necesidades de capacidad de procesamiento o ancho de banda.

NOTA: La forma de desplegar los distintos componentes de la aplicación en un entorno físico depende de múltiples factores. Para el caso de *BDA*, que puede considerarse una implementación

a mediana escala, la asignación de los componentes al entorno hardware se ha realizado tenido en cuenta factores como la cantidad de usuarios concurrentes; la velocidad y potencia de los distintos equipos; la velocidad y el ancho de banda de los enlaces de la red y estrategias de duplicación de componentes para obtener escalabilidad y una alta disponibilidad.

6.4.2.3 Vista Lógica

Descritas ya con un alto grado de abstracción las distintas capas de servicios que conforman el sistema, ahora se expone una visión conceptual del *CRM* de *BDA*. En Figura 6.18 se observa un diagrama simplificado, donde unicamente quedan expuestos aquellos elementos relevantes para el contexto actual.

En general, los componentes del sistema que se muestran en Figura 6.18 dependen de los componentes situados debajo de ellos en la infraestructura o de otros componentes transversales, a la vez que proporcionan respaldo a los componentes que están situados encima de ellos. Estas relaciones de dependencia y compatibilidad son un factor clave en el diseño de esta arquitectura lógica. En 6.4 se muestran las relaciones entre los distintos componentes que conforma la capa de lógica de negocio. A continuación, se ofrece una descripción de cada uno de los elementos reflejados en la figura.

- **HTML Client, Web Service Client**: Representan clientes consumidores de información que siguen el protocolo HTTP. Por ejemplo, un navegador web en un ordenador de sobremesa o un dispositivo móvil.
- **Framework**: Este elemento transversal representa el conjunto de componentes ofrecidos por el framework base sobre el cual está construido el *CRM* de *BDA*. Estos componentes están por lo general débilmente acoplados, lo que facilita su re-utilización, extensión y en definitiva constituyen una API. Por ejemplo, ofrece componentes e interfaces básicos o genéricos que permiten construir otros más complejos que se adecuen a las necesidades.
- **Dispatcher & Routers**: Parte del sistema encargada de interpretar una petición HTTP, inicializar el/los módulo/s correspondiente/s. Además, entre este elemento y los inferiores es donde se realizan labores de autorización y autenticación.
- **Presentation**: Componente o conjunto de componentes encargados de formatear y transformar los resultados para ser presentados al cliente.
- **Modules**: Representa aquellos componentes del sistema encargados de prestar servicio. Se trata de un conjunto de componentes software altamente cohesionados, y por lo general, fuertemente acoplados.
- **Data Access**: Parte del sistema encargada de ofrecer una abstracción sobre la capa de datos. Esta capa agrupa múltiples componentes que permiten interactuar con, por ejemplo, distintos motores de bases de datos.
- **Data, External Services**: Estos elementos representan todos los posibles orígenes de datos que alimentan al sistema. Aunque estos orígenes son principalmente motores de bases de

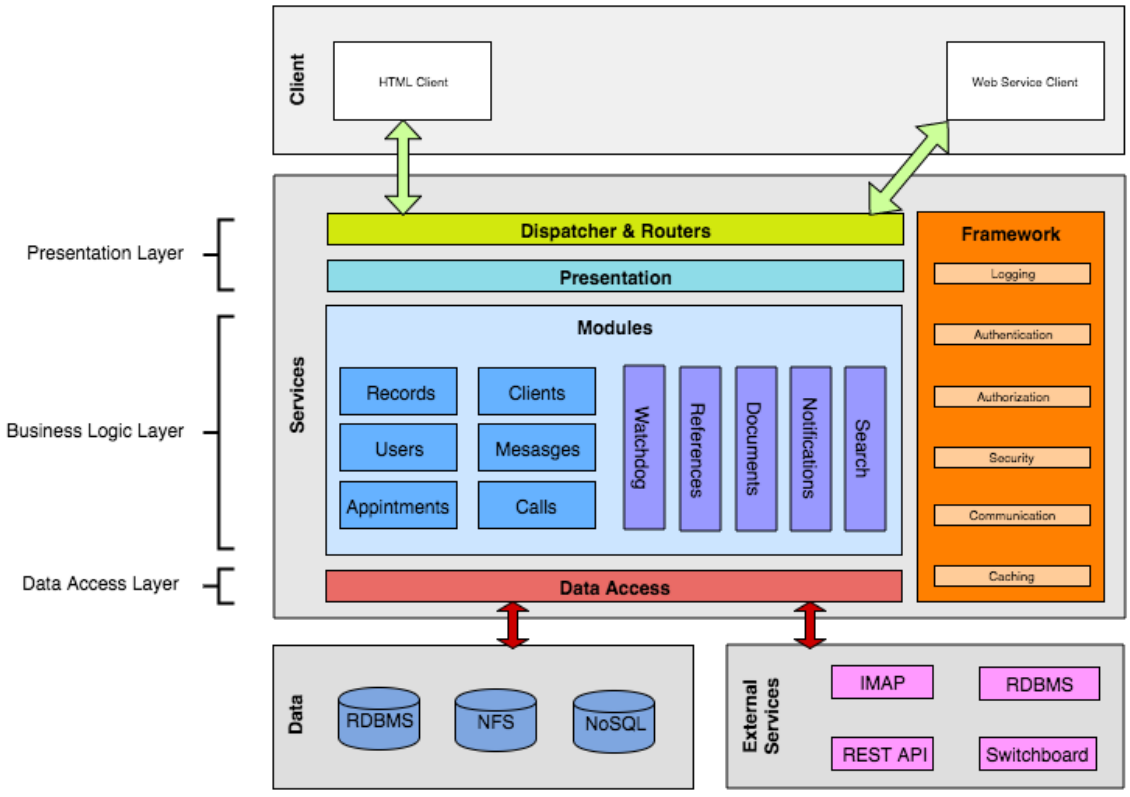


Figura 6.18: Vista lógica de la aplicación *CRM* de *BDA*.
Las tres grandes secciones representan los distintos niveles o tiers vistos anteriormente. Los componentes verticales indican transversalidad; utilizados por otros componentes del mismo nivel.

datos (relacionales y no relacionales), también es posible encontrar orígenes como centralitas telefónicas, servicios IMAP o incluso datos provenientes de servicios externos como Google.

Componente	**Depende de**	**Respalda el funcionamiento de**
Records	Clients, Appointments, Search, Watchdog, Documents, Notifications, Calls	Clients, Users, Calls
Users	Messages, Search, Watchdog, Notifications	Records, Clients, Watchdog
Appointments	Records, Clients, Watchdog	Records
Messages	Users, Records, References, Notifications	Users, Records
Search	Ninguno	Records, Clients, Users
Clients	Records, Watchdog, Calls	Records, Appointments, Calls
Watchdog	Users	Records, Users, Appointments, Messages
References	Ninguno	Messages
Documents	Ninguno	Records
Calls	Clients, Records	Records, Clients

Tabla 6.4: Relación de dependencias entre componentes del CRM de BDA

6.5 Arquitectura de Procesos De Negocio

La gestión de procesos de negocio (o *BPM*), proporcionan una forma de monitorizar y mejorar la eficiencia del negocio. *BPM* recoleta múltiples datos de la organización provenientes de distintas aplicaciones de la organización, y luego realiza dos cosas: 1) rastrea cómo la información se está utilizando para así completar el negocio, y en función de esto localizar con precisión los procesos de negocio existentes y así lograr comprenderlos; 2) realiza un seguimiento de los flujos de información de las operaciones, para así asegurar que un proceso se ejecuta de forma correcta. *[ZhenyuFang2010]*

En este capítulo se ofrece una visión dinámica del sistema desde el punto de vista organizativo, con un elevado nivel de abstracción y sin ahondar en detalles internos de los sistemas software involucrados. Los resultados aquí expuestos son el resultado de un estudio completamente independiente y paralelo a este documento, realizado durante la asignatura de "Investigación e Innovación" [1] del Máster en Ingeniería Informática. Los detalles de este estudio se pueden consultar en el apartado de *Anexos*; se invitar al lector a consultar el documento titulado "Minería de Procesos".

6.5.1 BDA y la Minería de Procesos

BDA no contaba con un modelo de procesos formalizado y bien definido en cuanto a la gestión de los casos jurídicos. Esta gestión suele ser muy personalizada, guiada por circunstancias y por unas directrices básicas de conocimiento común en la organización.

El *CRM* de *BDA* fue concebido desde sus inicios (como uno de sus requisitos) con idea de monitorizar al máximo posible la actividad del personal, por ello es que una de sus piezas fundamentales es el denominado "Watchdog" o "perro guardián": un componente software que actúa a bajo nivel observando, interceptando y registrando la actividad de la base de datos subyacente; una especie de *Sniffer* para la capa de acceso a datos. Este componente software resulta ser un elemento interesante desde la perspectiva de la minería de procesos, pues ofrece un registro de eventos ajustados a la realidad de la organización, lo que lo convierte en un elemento apto para el estudio en este sentido.

Una vez se aplicaron las técnicas de minería de procesos, se obtuvieron los resultados generales observables en Figura 6.19, éstos permitieron mejorar y automatizar determinados aspectos del sistema, aspectos que de otro modo habrían sido, si no imposibles, difíciles abordar de otra manera. Este modelo refleja, de forma fiel y ajustada a la realidad el proceso general utilizado por la organización en lo referido a la gestión de sus casos jurídicos. Es importante señalar que, la obtención de este modelo únicamente fue posible una vez se desplegó el *CRM* de *BDA* en un entorno de producción. Y que, anterior a esta situación no se contaba con ningún modelo formalizado, bien definido y ajustado a la realidad.

[1] Ficha de la asignatura "Investigación e Innovación" http://www.unavarra.es/ficha-asignaturaDOA?languageId=100000&codPlan=721&codAsig=73298

6.5.1.1 Sobre el Modelo

En el modelo de Figura 6.19, cada recuadro representa una actividad. Cada actividad viene acompañada de un nombre y un número que indica la frecuencia relativa con que esta ocurre; un valor igual a 1 indica que dicha actividad ocurre siempre en todas las ejecuciones del proceso. Por otro lado, el grosor de los arcos entre dos actividades indica cuán a menudo una actividad es ejecutada después de otra. Por ejemplo, la actividad "file-upload" tiene un arco consigo mismo de un grosor considerable, de esto se intuye que la actividad de subida de documentos es secuencial en el caso de las subidas en masa.

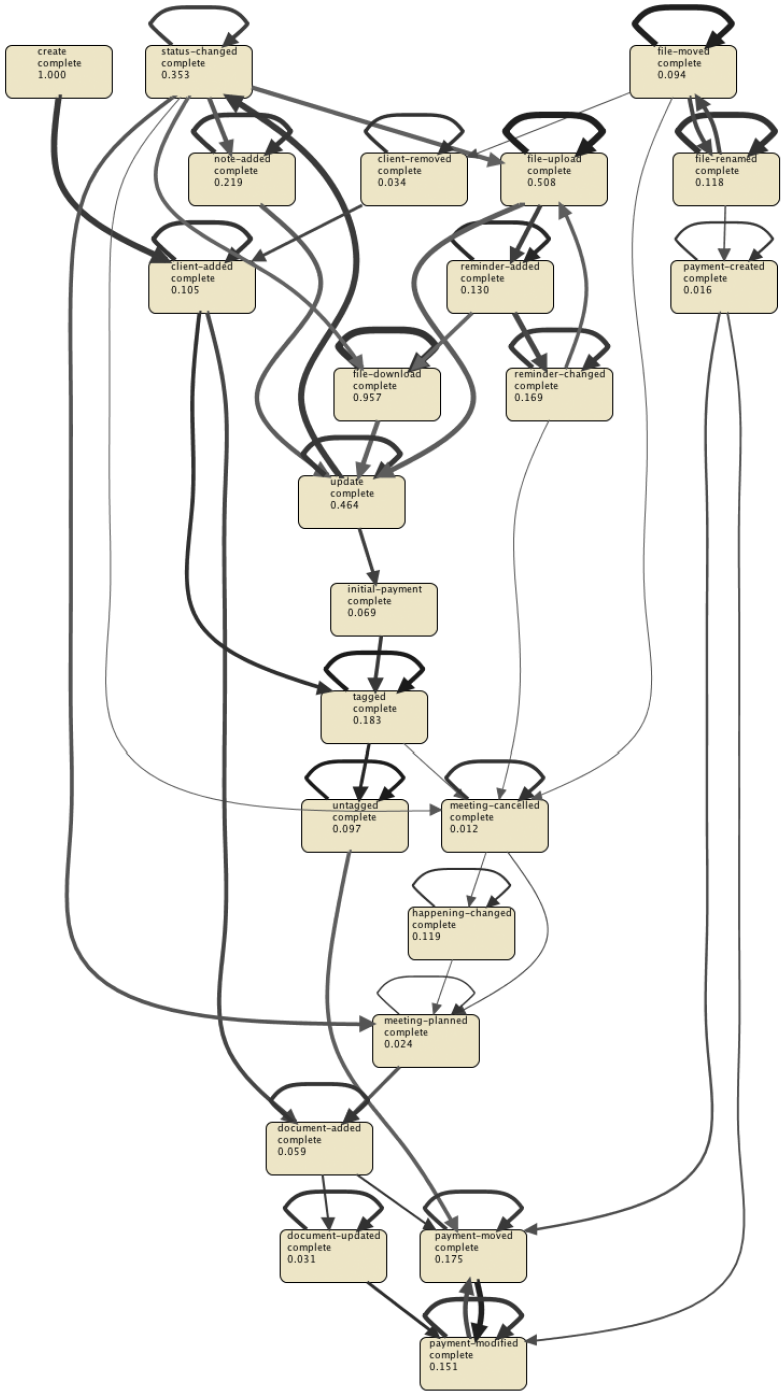


Figura 6.19: Modelo de proceso descubierto aplicando técnicas de minería de procesos. Consultar anexo "Minería de Procesos" para una visión pormenorizada. Ahí se explican, detallan y justifican una serie de cuestiones relevantes para la Arquitectura de Procesos de Negocio.

6.6 Arquitectura de Datos

La arquitectura de datos define como los datos son almacenados, gestionados y utilizados en el sistema *CRM* de *BDA*. En este capítulo se describe, entre otras cosas, las políticas que gobiernan a nivel organizacional el uso y tratamiento de estos datos; descripción de componentes software; representación y comunicación con sistemas externos.

6.6.1 Arquitectura de Datos Previa

La situación inicial de *BDA* en cuanto al tratamiento de los datos resulta más bien escueta. No existe una política ni arquitectura bien definida que ofrezca una visión del sistema en este sentido. El funcionamiento a nivel organizativo consiste básicamente en:

- Cumplimentar una serie de hojas de cálculo provenientes de múltiples departamentos internos.
- Una vez por semana se combinan todas estas hojas y se produce una única hoja de cálculo con la información de todas ellas. Esta tarea es completamente manual y realizada por una persona física.
- Una vez generado la hoja de cálculo anterior se convierte, mediante un proceso automatizado, en múltiples tablas relacionales que alimentan una interfaz web.

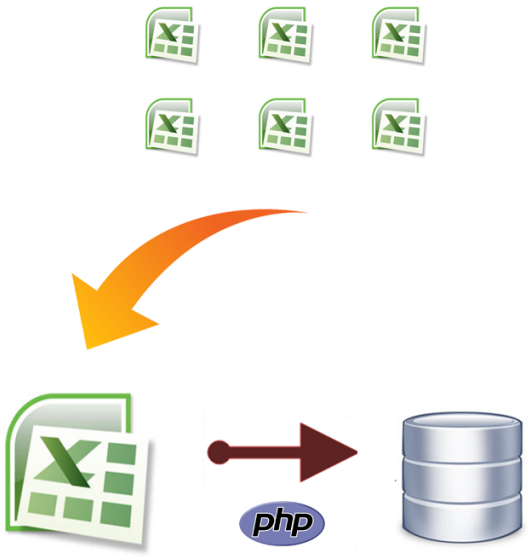


Figura 6.20: Esquematización proceso de importación datos sistema *CRM* de *BDA* anterior.

En relación al almacenamiento masivo documental, estos datos se encuentran en un servicio externo denominado “Flexxible Desktop”, un entorno virtual de escritorios remotos. Ofrece un entorno privado basado en MS Windows, con su propia red y ordenadores virtuales. Dentro de esta

red virtual se cuenta con un servidor Windows que comparte un directorio de más de 6TB sobre el cual se almacena toda la documentación.

Las deficiencias que se producen producto de este modelo de funcionamiento son varias: inconsistencia en los datos producto de fallos humanos, redundancia de información causadas por una falta de normalización (en muchos casos injustificada e innecesaria) de los datos; múltiples fallos en el proceso conversión automatizado; y graves problemas en términos de seguridad como, por ejemplo, exposición de información sensible sobre canales no autorizados.

6.6.2 Arquitectura de Datos Actual

6.6.2.1 Políticas de Datos

La organización *BDA* exige una serie de controles para gestionar de forma apropiada el riesgo sobre la confidencialidad, integridad y disponibilidad de los datos que ésta gestiona. A continuación, algunas de la políticas internas sobre el tratamiento de los datos impuestas por la organización. A estas "normas" internas se pueden añadir controles adicionales en función de las leyes aplicables:

- No existe la eliminación física de datos relacionales. En su lugar se ha de implementar mecanismos de *soft-delete*.
- De igual modo, el almacenamiento documental masivo es incremental. Ningún documento es eliminado físicamente. En su lugar se ha de implementar un sistema de "papelera" que permita incluso recuperar documentos eliminados. No es posible "vaciar" esta papelera virtual.
- En términos de seguridad, se definen tres niveles o clasificaciones de los datos dependiendo del uso y tratamiento esperado de los mismos:
    - Datos Públicos: Datos considerados de uso público a nivel organizacional, y que cuando se usan de forma apropiada, tienen poco o ningún efecto adverso sobre las operaciones, activos, reputación de la organización o sobre las obligaciones de la organización en términos de privacidad.
    - Datos Internos: Datos no destinados a un uso fuera de la organización, y que en caso de ser desvelados podrían tener un efecto adverso sobre las operaciones, activos, reputación de la organización o sobre las obligaciones de la organización en términos de privacidad.
    - Datos Sensibles: Datos destinados a un uso restringido dentro de la propia organización, y que en caso de ser desvelados es espera que produzcan un efecto adverso sobre las operaciones, activos, reputación de la organización o sobre las obligaciones de la organización en términos de privacidad.
- Se debe tener un control sobre el acceso o modificación de datos considerada sensible, esto en cumplimiento de los marcos legales correspondientes.

6.6.2.2 Requisitos de Datos

Los requisitos de datos están guiados por una serie de requerimientos funcionales y no funcionales. Algunos ejemplos de estos son los siguientes:

- Estadísticas e Informes: Este tipo de consultas involucran la extracción, relación y agregación de datos de una o más tablas. Los informes que el sistema ofrece están siempre bajo cambios constantes y nuevos informes aparecer de forma frecuente.
- Transacciones complejas: El sistema debe soportar de forma eficiente un gran volumen de transacciones, además estas transacciones podrían involucrar elementos dispersos en distintas áreas del sistema. Por ejemplo, documentos binarios en almacenamientos masivos.
- Resúmenes: Algunas estadísticas e informes pueden agregar grandes volúmenes de datos provenientes de múltiples tablas u orígenes de datos. El cálculo de esta información puede ser computacionalmente intensiva, por lo que es necesario idear soluciones de almacenamiento temporal o caches.

6.6.2.3 Requisitos Tecnológicos

Los requisitos tecnológicos son los sugeridos o impuestos por los marcos de desarrollo de la organización, *BDA*. En el caso de *BDA*, inicialmente no existía ninguna imposición ni sugerencia al respecto, aunque se muestra abierta y flexible en este sentido. Por este motivo, las decisiones en cuanto a qué tecnologías adoptar recaen como una de mis funciones. A continuación, algunas de las tecnologías adoptadas:

- Plataforma web dinámica basada en *PHP*.
- Framework CakePHP.
- MySQL como sistema gestor de bases de datos relacionales.
- MongoDB como base de datos documental.

Las elecciones de estas tecnologías quedan envueltas por un proceso previo de análisis y comparativa entre distintas alternativas. Si bien este proceso se escapa al alcance de este documento, es importante señalar que el factor determinante en este proceso resulta ser en gran medida el "nivel de conocimiento" y "experiencia previa" sobre cada una de las plataformas o tecnologías implicadas; lo que ayuda a reducir de forma considerable el riesgo de fracaso del proyecto por este tipo de amenazas.

6.6.2.4 Componentes de Datos

Los componentes de datos proporcionan encapsulamiento y un mecanismo de comunicación con los sistemas de almacenamiento para recuperar o modificar información. En la arquitectura de datos del *CRM* de *BDA*, los componentes de datos quedan implementados como objetos tipo Entidad

ORM. En el caso de datos no relacionales o no normalizados (como, por ejemplo, datos provenientes de una base de datos MongoDB), se utilizan implementaciones *OOP* ad-hoc según sea el caso.

Los componentes de datos representan a un alto nivel los datos provenientes de (generalmente) una base de datos y añaden comportamiento a tales datos. Así, en lugar de implementar lógicas ligadas a la base de datos, la aplicación simplemente utiliza interfaces de la capa de acceso a datos para acceder a los datos.

El acceso a la base de datos se hace por medio de dos objetos primarios. El primero son repositorios u objetos tabla. Estos objetos proporcionan acceso a colecciones de datos, permiten guardar nuevos datos, modificar y borrar existentes, definir relaciones y realizar operaciones en masa. El segundo tipo de objeto son las entidades, representan registros/filas individuales y permiten definir funcionalidad y comportamiento a nivel de registro/fila.

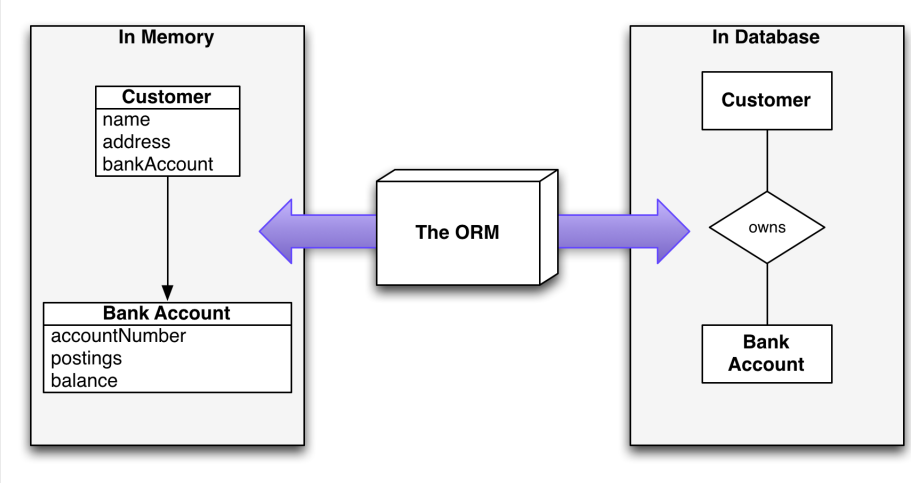


Figura 6.21: Esquematización funcionamiento *ORM*. Fuente: ActiveAndroid Guide.

El *ORM* utilizado en el *CRM* de *BDA* es ofrecido por el framework de CakePHP. Éste toma ideas y conceptos de los modelos *ActiveRecord* y *Datamapper*. Aspira a crear una implementación híbrida que combine aspectos de los dos modelos para crear un *ORM* rápido y fácil de usar.

6.6.2.5 Modelado Entity-Attribute-Value

La forma convencional de representar atributos para una clase en un modelo relacional es mediante columnas en una tabla, una columna por atributo. Este enfoque, conocido como modelado por filas, es adecuado para clases con un número fijo de atributos, donde la mayoría o todos los atributos tienen valores para una determinada instancia. Sin embargo, la representación de atributos como columnas no funciona bien para clases con un número potencialmente grande de atributos, donde una instancia dada tendría sólo unos pocos atributos no vacíos. Los atributos pueden ser vacíos por múltiples razones, por ejemplo, por ser desconocidos o inaplicables.

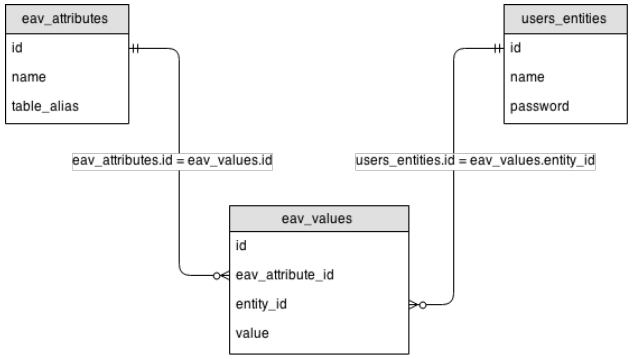


Figura 6.22: Modelo EAV.
Una entidad puede definir una serie de atributos y valores asociados a estos últimos.

Dado que los motores de base de datos relacionales estándar están limitados a unas 1024 columnas por tabla (o menos), no es posible diseñar una estrategia sistemática y racional para particionar el universo de atributos a través de cien o más tablas. Además, los parámetros aumentan continuamente a medida que avanza el conocimiento sobre las clases implicadas, requiriendo continuas modificaciones de los esquemas y de las interfaces de usuario. El diseño *EAV* es una generalización del modelado por filas, en donde una única tabla (o conjunto de tablas) es utilizada para almacenar todos los datos ligados a un atributo en particular para una entidad concreta.

En el caso de *BDA* se ha decidido emplear el modelo EAV en determinadas partes del sistema, en concreto sobre partes del sistema que se ajusten a las siguientes circunstancias:

- No es posible modelar esta información sobre motores de bases de datos no relacionales.
- El conjunto requerido de atributos es arbitrario y desconocido.
- Se espera que los esquemas de estas tablas cambien con regularidad.
- La información asociada a estos atributos no es relevante para la localización/búsqueda de entidades.

6.6.2.6 Modelo Lógico de Datos

La clase principal en el modelo de datos del *CRM* de *BDA* (Figura 6.23) es “Case”, que:

- Representa los expedientes (o casos) jurídicos;
- cada expediente concreto (objeto) contiene todos los componentes necesarios para utilizarlo, datos, metadatos y ficheros/documentos;
- tiene enlaces con múltiples objetos tipo “Client”, y con otros objetos de tipo “Case” cuando se trata de un expediente de tipo “colectivo”;

En cuanto al diagrama es necesario aclarar las siguientes cuestiones semánticas y de notación:

- Clases abstractas: todas las clases abstractas se identifican por sus nombres en *cursiva*.

- Atributos EAV: las clases marcada en color magenta representan agrupaciones lógicas de atributos virtuales. De este modo, por ejemplo, la clase "User" hereda siempre un conjunto fijo de atributos. Estas agrupaciones se conocen bajo el nombre de "bundle", véase el documento adjunto en la sección de anexos (*EAV Plugin*) ahí se explica este concepto en detalle. Por motivos de espacio se han omitido los atributos de estas clases, para conocer el listado completo de atributos de cada clase consulte en la sección de anexos, el apartado *Bundles EAV*.
- Generalización dinámica: el trapezoide representa una generalización multiplexada. El concepto (y el símbolo) es el mismo utilizado en electrónica: en función de una(s) entrada (línea punteada izquierda) se conecta una de las entradas (abajo, parte más ancha) a la salida (arriba, parte más angosta). De este modo, en el diagrama, un objeto de la clase "Case" generaliza una u otra clase (*GastosHipoteca*, *GastosHipotecaYCS* o *CSuelo*) en función del valor del atributo "caseType" de cada objeto.

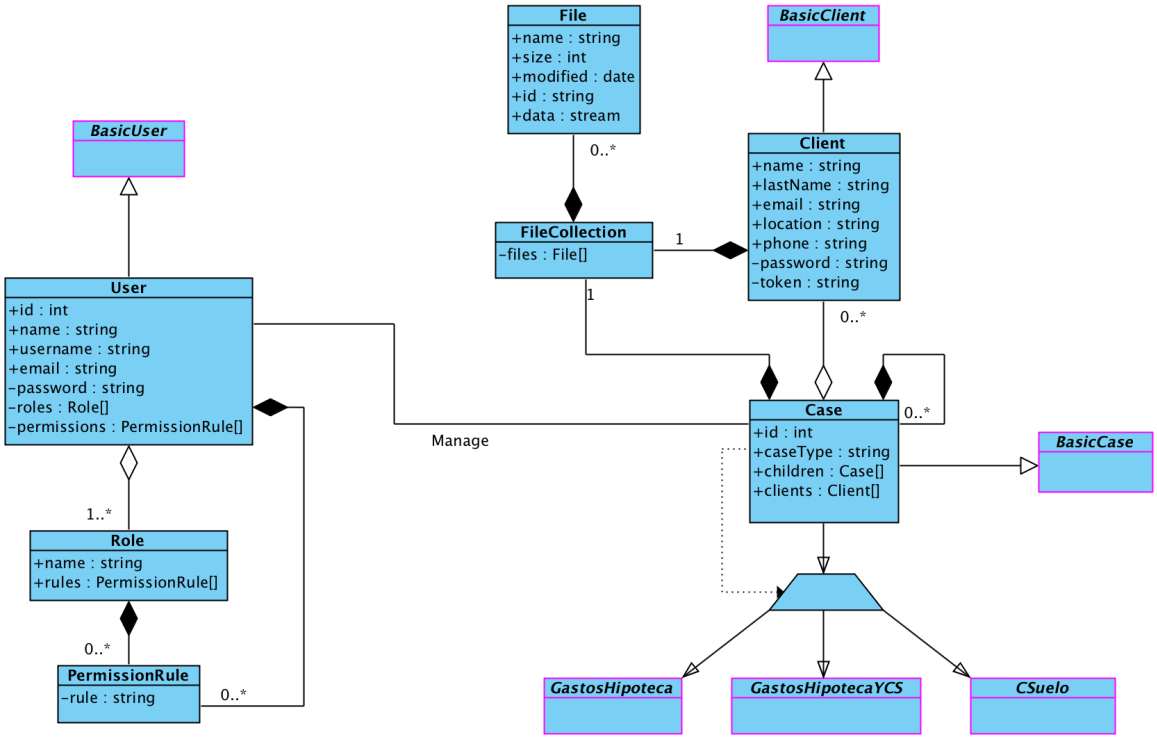


Figura 6.23: Diagrama UML simplificado del modelo de datos del *CRM* de *BDA*.
Por consideraciones de espacio se han omitido algunas entidades y sus asociaciones, quedando únicamente expuestos aquellos elementos más importantes desde la perspectiva de los datos.

En la sección de *Anexos*, documento *Diagrama de Datos EAV*, se ofrece una representación gráfica completa de todos las propiedades virtuales y no virtuales de cada entidad y "bundle", junto con sus tipos de datos y sus respectivas generalizaciones.

6.6.2.7 Modelo Físico Base de Datos CRM

En la sección se apéndices (*Modelo Relacional*) se ofrece un diagrama que refleja los elementos principales del sistema *CRM* de *BDA*. Dicho diagrama representa una visión física normalizada de la base de datos relacional primaria utilizada por el *CRM* de *BDA*.

6.6.2.8 Respaldo y Recuperación de Datos

El sistema *CRM* de *BDA* cuenta y requiere de procedimientos automatizados de respaldo de datos para garantizar que la información pueda ser recuperada en caso de catástrofes. A continuación de describen algunas de las políticas adoptadas y configuraciones en relación a los dos almacenes de datos principales, bases de datos y documentos.

Bases de Datos

Las bases de datos principales se replican de forma síncrona en una configuración maestro- maestro bajo una configuración en cluster. Esta configuración garantiza que, ante el fallo de una maquina física, ya sea por interrupción del servicio o fallo de la máquina física, el sistema de bases de datos sigue operando con normalidad sin interrupción del servicio. Además de contar con este replicado en tiempo real, se realizan (una vez por día) copias completas de forma automatizada, estas copias son almacenadas durante tres meses antes de ser eliminadas.

Por otro lado, se cuenta con un replicado externo al cluster mediante una configuración maestro-esclavo retardada, este retardo ofrece un margen de maniobra en caso de un desastre de tipo lógico, por ejemplo, eliminación accidental de datos por un mal funcionamiento del software o un error humano.

Almacenamiento Documental

En cuanto al almacenamiento masivo de documentos, se cuenta con un soporte físico redundado (RAID-10) que asegura la consistencia y disponibilidad de los datos en caso de un fallo de los soportes físicos. Además, a nivel lógico, se realiza una copia semisíncrona de todos los documentos en un almacenamiento pasivo de similares características. Estas máquinas están configuradas con IPs virtuales [1] en modo maestro-esclavo con "failover" automático, es decir, en caso de caer el dispositivo de almacenamiento principal (maestro) se asciende un esclavo como nuevo maestro de forma automática.

[1] https://en.wikipedia.org/wiki/Virtual_IP_address

CAPÍTULO 7

Conclusiones

El objetivo principal de este documento, como se explica al inicio, se puede resumir demostrar la adecuada aplicación y entendimiento de los conocimientos adquiridos durante el Máster de Ingeniería Informática, mediante la presentación de un proyecto que resuelve problemas y necesidades reales de la sociedad y el mercado.

A lo largo de este documento se ha intentado reflejar de forma fiel la realidad de una organización en el ámbito tecnológico; mediante la exposición de su situación previa y posterior a la implantación de un renovado sistema de información personalizado. La implantación de un sistema de este estilo requirió de unos conocimientos más extensos que los manifestados en este documento, esto se debe a que los alcances de los contenidos aquí expuestos se han limitado de forma intencionada a fin de acomodarse a los requisitos y competencias específicas del Máster en Ingeniería Informática; quedando excluido, por ejemplo, todos aquellos temas ligados a la Ingeniería de Software.

Participar en este proyecto ha supuesto alcanzar un alto grado de satisfacción profesional y personal, aunque ha sido todo un reto a distintos niveles durante todas sus fases. Por una parte, el aprendizaje de multitud de nuevas tecnologías y, por otro, elementos de índole laboral como las constantes presiones en la consecución de objetivos y resultados dentro unos marcos temporales, a mi juicio, algo ajustados.

Este proyecto me ha permitido profundizar y aplicar conocimientos adquiridos a lo largo del Máster en Ingeniería Informática. Por ejemplo, conocimientos relacionados con sistemas distribuidos y de virtualización, redes y teoría de colas; sobre gestión y dirección de proyectos, han resultado ser de especial relevancia en la práctica durante la ejecución de este proyecto. Aunque también es necesario señalar que, como cabría de esperar, fue necesario ahondar en algunas materias imprescindibles para lograr unos niveles de calidad adecuados. Situación que me llevó en varias ocasiones a realizar unas profundas sesiones de auto-formación sobre temas o tecnologías específicas que se escapaban alcance del Máster en cuestión.

El sistema de información implantado en *BDA* está siempre en constante evolución. Incluso al momento de redactar estos párrafos existen ya grandes planes de expansión y diversificación del sistema, y que por motivos temporales resulta imposible ofrecer una visión “actualizada” en este

documento del sistema en cuestión. La implantación de este sistema no significa que todos los problemas de la organización quedan ya resueltos y terminados, al contrario, este sistema ofrece una visión única a la organización y proporciona a la dirección información oportuna y precisa para apoyar la toma decisiones. Personalmente pienso que el éxito o fracaso no depende de si se cuenta con un sistema de información adecuado, sino de las decisiones que se adopten utilizando la información que el sistema o sus derivados puedan ofrecer.

CAPÍTULO 8

Apéndices

8.1 Plan de Gestión

- **Objetivo**: Identificar el marco de condiciones, y necesidades vigentes y proyectadas, de referencia para el diseño, el desarrollo e implantación del *CMR* de *BDA*.
- **Actividades**:
  - Diagnóstico de situación:
    - Diagnóstico de organización
      - Modelo organizativo, eficacia y conflictos subyacentes,
      - Eficacia de la implantación de responsabilidades y funciones.
      - Eficacia de la relación entre competencias y cargos.
    - Diagnóstico de procesos y procedimientos.
      - Estado de definición de procesos claves (documentada / no documentada)
      - Estado de conocimiento y aplicación de procedimientos claves.
      - Nivel de eficacia de los procesos e identificación de conflictos.
    - Diagnóstico de los recursos informáticos disponibles
      - Nivel de obsolescencia del hardware – Identificación de conflictos.
      - Nivel de eficacia del software con relación a los procesos.
      - Adecuación de interfaces e informes a los usos previstos.
    - Evaluación de plataformas externas que utiliza la empresa.
    - Declaración de conflictos y necesidades de los usuarios.

    - Adecuación tecnológica de la empresa respecto de su entorno y tendencias

---

- **Objetivo**: Formular objetivos y requisitos aplicables al proyecto *CRM* de *BDA*.
- **Actividades**:
  - Definición de objetivos de interacción, funcionalidades e interfaces gráficas con orientación al uso intuitivo del sistema.
  - Definición del concepto y características fundamentales para el diseño.
  - Definición de arquitecturas de soporte; de código, física, lógica y de datos.

- **Objetivo**: Prevenir errores o déficit del diseño.
- **Actividades**:
  - Evaluación de la capacidad del diseño para satisfacer los requisitos (funcionales y no funcionales).
    - Contrastación de requisitos y elementos de diseño.
    - Usabilidad de interfaces gráficas.
  - Depuración y/o mejora del diseño.

- **Objetivos**: Codificación
- **Actividades**:
  - Desarrollo de interfaces gráficas (Capa de usuario), frontend.
  - Desarrollo de capa de servicios, backend.

- **Objetivos**: Definir y cumplir la implantación eficaz del proyecto *CRM* de *BDA*
- **Actividades**:
  - Plan de implantación
    - Definición de áreas pilotos de implantación.
    - Definir ensayos de sistemas, ciclos de prueba, métodos y medios de evaluación.
    - Plan piloto de formación a usuarios.
  - Despliegue de pruebas beta cerradas.
    - Formación de usuarios.
    - Despliegue software beta.

    - Uso y evaluación del sistema en fase beta.
- Mejoras al sistema antes de despliegue en producción.
- Despliegue en un entorno de producción.
    - Formación del equipo de soporte (niveles 1, 2 y 3).
    - Formación a usuarios del sistema.
    - Despliegue del sistema en entorno de producción.
    - Implantación de rutinas de mantenimiento, evaluación y mejora continua.

8.2 Agile Aplicado a BDA

Una vez se establecieron una serie de objetivos para el proyecto *CRM* de *BDA*, fue necesario establecer un método de trabajo que garantizara unos mínimos en cuanto a calidad y fiabilidad del producto final de desarrollar. Analizadas las características del proyecto y consideradas diversas metodologías, se optó por adoptar una metodología de tipo "ágil" o *LEAN* de desarrollo frente a otras metodologías tradicionales. Las motivaciones de esta decisión fueron varias, aunque es necesario indicar que todo este proceso de decisión estuvo guiado por las siguientes cuestiones:

1. **Objetivos de proyecto y requisitos**. El cliente presenta dificultades para expresar y definir requisitos. Un elevado grado de incertidumbre en cuanto a la definición de los requisitos; cambian con regularidad a lo largo del proyecto.
2. **Disponibilidad del cliente**. El cliente se muestra colaborativo y con un alto nivel de implicación en la consecución de objetivos.
3. **Equipo de trabajo**. Es pequeño y poco experimentado, tanto en tecnologías como en metodologías de trabajo.

8.2.1 Uso de Metodologías Ágiles o LEAN

El hecho de que las metodologías tradicionales estén ideadas para un uso exhaustivo de documentación durante todo el ciclo y que se centren en cumplir con un plan de proyecto, las hacen menos atractivas o apropiadas para el escenario de *BDA*. En contraste, las metodologías ágiles de desarrollo ponen vital importancia en la capacidad de respuesta a los cambios, la confianza en las habilidades del equipo y al mantener una buena relación con el cliente. Algunos de sus principios más destacados son: el cliente es la mayor prioridad; los cambios de los requisitos son bienvenidos; entrega continua; desarrolladores colaboran de forma diaria con personas que entienden el negocio; conversaciones cara a cara; progresos en función de software funcional; simplicidad y equipos auto-organizados.

Las ideas de solución son simples hipótesis y deben ensayarse para asegurar que sirven y funcionan correctamente; el fracaso de una idea no es tal sino aprendizaje y punto de partida para nuevas ideas y experimentos (Pivotar); una solución es buena sólo si los usuarios la usan y les es útil.

Lo esencial de nuestra metodología consiste en:

- Identificar síntomas de problemas.
- Analizar síntomas para establecer causas raíces o problemas reales.
- Proponer soluciones (hipótesis).
- Analizar pros y contras de las ideas de solución (hipótesis) con los usuarios.
- Formular mejoras al concepto.
- Generar desarrollos de ensayo de conceptos (Betas).
- Ensayar y mejorar soluciones beta.
- Perfeccionar soluciones definitivas a partir de ensayos beta.
- Implantar soluciones definitivas.
- Formular mejoras al concepto

8.2.2 Marco / eXtreme Programing (XP)

Una vez establecido el tipo de metodología a emplear, fue necesario concretar y adoptar un marco concreto: *eXtreme Programming* o *XP*. La adopción de este marco de trabajo se hace con la idea de establecer unos mínimos en cuanto a la gestión del proyecto y arraigar ciertas costumbres en el equipo de desarrollo.

Lo que se pretende no es cumplir de forma rigurosa y estricta absolutamente todos los principios y filosofías que el marco pueda establecer (como, por ejemplo, la propiedad compartida del código), sino que esta elección se hace con la idea de que el marco sea una mera guía de soporte al desarrollo del proyecto. De este modo se ha decidido considerar aquellos principios y filosofías más beneficiosos para el desarrollo del proyecto como, por ejemplo, la integración continua, la ejecución automatizada de pruebas y emplear estándares de codificación.

Por otro lado, es importante señalar que, incluso algunos principios propuestos en el propio *Manifiesto Ágil* han sido descartados. Por ejemplo, dicho manifiesto propone valorar más el software que funcione que la documentación exhaustiva y su integración en el proceso de desarrollo. Si bien este principio podría tener cierto valor en contextos determinados, resulta ser un elemento extremadamente negativo para un proyecto en donde el equipo de desarrollo se encuentra segmentado a términos geográficos y no existe una comunicación fluida y constante entre todos los miembros. Si bien el equipo de desarrollo en su totalidad son menos de 10 personas la mitad de estos se encuentran en provincias separadas, luego la correcta documentación de los desarrollos es una actividad esencial para transmitir conocimiento y el funcionamiento del sistema al resto del equipo.

8.2.3 Otras Metodologías

Otra metodología ágil evaluada en esta fase fue *Scrum*, pero a pesar de compartir muchas ideas fue descartado en beneficio de *eXtreme Programming*, por resultar complejo de planificar por un Scrum Master en un entorno como el de *BDA*; un proyecto en el cual el cliente manifiesta una incapacidad para transmitir de forma clara los requisitos, por lo resulta complicado intentar estructurar y organizar de forma precisa cada aspecto del mismo. Los cambios constantes, las entregas continuas y la incertidumbre en cuanto a la naturaleza precisa del producto acabado hacen que el ciclo de vida del proyecto sea demasiado intensivo para todos los involucrados. Además, las reuniones diarias de *Scrum* y las constantes revisiones requieren de unos recursos considerables.

Scrum y eXtreme Programming son en realidad muy similares. Tanto que quizás sea difícil diferenciar a simple vista si un equipo sigue una u otra metodología. Las diferencias son más bien sutiles, pero importantes. A continuación, un listado de cuatro de estas diferencias:

- Los equipos Scrum generalmente trabajan en iteraciones denominadas "sprints", que van típicamente desde una semana hasta un mes. Los equipos *XP* trabajan por iteraciones de entre una a dos semanas.
- Los equipos Scrum no permiten modificaciones durante un sprint. Una vez concluye la reunión de planificación del sprint y se establece un conjunto de elementos a entregar al finalizar el sprint, dicho conjunto de elementos se mantiene inalterado hasta el término del mismo. Los equipos de *XP* están mucho más abiertos a los cambios una vez iniciado un sprint. Siempre y cuando el equipo no haya comenzado ya a trabajar en una característica, una nueva característica de tamaño equivalente se puede intercambiar en la iteración en curso a cambio de la característica no iniciada.
- Los equipos *XP* trabajan bajo un orden de prioridades. Las características a desarrollar son priorizadas por el cliente (o en Scrum, "Product Owner") y el equipo debe trabajar en ellas en ese orden. En contraste, en Scrum es el "Product Owner" quien prioriza el backlog, pero es el equipo el que determina la secuencia en que se desarrollaran los items del backlog. Esto significa que en Scrum el equipo puede decidir desarrollar antes un item con menor prioridad.
- Scrum no establece ninguna práctica de ingeniería, *XP* en cambio sí lo hace.

El objetivo de estas comparativas no es el de determinar cual metodología es mejor que otra, sino explorar las diferencias entre ellas y justificar la decisión de adoptar una u otra bajo un escenario concreto, *BDA*. Ambas metodologías se ajustan a las principales filosofías del denominado "Manifiesto Ágil", el cual intenta ofrecer al producto el mayor valor posible. Las diferencias entre estas metodologías son el resultado de intentar mantener los principios ágiles en contextos radicalmente diferentes.

Por una lado, Scrum es más adecuado para los equipos que pueden dedicar su tiempo a un proyecto o producto. Establece más bien una estructura que permite aumentar la productividad del equipo manteniendo planificaciones y comunicaciones constantes, a la vez que otorga un alto grado de libertad entre el equipo a la hora de tomar decisiones de ingeniería. Por otro lado, XP añade una capa más de sofisticación, presta especial atención a la calidad de los desarrollos mediante la

"imposición" de un conjunto de buenas prácticas reconocidas y centradas en mantener el código software limpio, legible y estable.

Como se comentaba anteriormente y como se observa en esta breve comparativa. Algunos elementos pueden intercambiarse, descartarse o modificarse en función del contexto y de las cualidades del proyecto (como así se ha hecho en el caso de *BDA*). En ocasiones esto puede llevar a situaciones de prueba y error para conseguirlo. Sin embargo, siempre y cuando el hilo conductor de este proceso sean los "Principios Ágiles" se puede estar seguro de que el camino es el adecuado. Y es que (en mi opinión) un proyecto exitoso depende en definitiva de la madurez y la dedicación de todos sus participantes, y de su capacidad para mantener una una comunicación y compromiso constante sobre cada retraso o revisión.

8.3 eXtreme Programming

eXtreme Programming es una metodología de desarrollo de software diseñada para mejorar la calidad del software y su capacidad para adaptarse adecuadamente a las necesidades cambiantes del cliente. Durante la mitad y finales de los noventa, el ingeniero de software Ken Beck desarrolló la metodología de programación extrema. En octubre de 1999, publicó "Extreme Programming Explained", detallando todo el método para otros, y poco después se publicó un sitio web oficial.

Similar a otras metodologías ágiles de desarrollo, eXtreme Programming se enfoca en ofrecer pequeños entregables de forma iterativa y con regularidad, lo que permite tanto a los miembros del equipo como al cliente evaluar y revisar el progreso del proyecto durante todo el ciclo de desarrollo.

En este apartado se intentará explicar qué es exactamente eXtremme Programming, cómo funciona, sus principios y valores. La mayoría de lo aquí expuesto es un extracto de *[KenBeck1999]*, se invita al lector a consultar la sección bibliográfica para una mayor información.

8.3.1 Valores Extremos

Estos cinco valores fundamentales constituyen la base sobre la que se construye todo el paradigma de la eXtreme Programming, de forma que las personas involucradas en el proyecto se sientan cómodos con la dirección que está tomando el proyecto y entender su retroalimentación personal.

- **Simplicidad**: Haremos lo que se necesita y lo que pedimos, pero no más. Esto maximizará el valor creado para la inversión realizada hasta la fecha. Se realizarán pequeños y simples pasos en los objetivos y se solventarán los fracasos a medida que sucedan. Crearemos algo de lo que estamos orgullosos y lo mantendremos al largo plazo por unos costes razonables.
- **Comunicación**: Todo el mundo forma parte del equipo y nos comunicamos cara a cara todos los días. Vamos a trabajar juntos en todo, desde los requisitos hasta el código. Vamos a crear la mejor solución posible a nuestro problema de forma conjunta.
- **Retroalimentación**: Tomaremos en serio todo compromiso de iteración entregando software funcional. Enseñaremos nuestro software lo antes posible y realizaremos los cambios

necesarios de la mejor manera posible. Vamos a hablar sobre el proyecto y adaptar nuestro proceso a él, no al revés.

- **Respeto**: Todos dan y sienten el respeto que merecen como un miembro valioso del equipo. Todo el mundo aporta valor incluso si es simplemente entusiasmo. Los desarrolladores respetan la experiencia de los clientes y viceversa. La gerencia respeta nuestro derecho a aceptar responsabilidad y recibir autoridad sobre nuestro propio trabajo.
- **Valor**: Vamos a decir la verdad sobre el progreso y las estimaciones. No pondremos excusas para fracasar porque planeamos tener éxito. No tememos nada porque nadie trabaja solo. Nos adaptamos a los cambios cuando ocurran.

8.3.2 Reglas Extremas

Inicialmente publicado por Don Wells en 1999, propietario del sitio web de eXtreme Programming, este conjunto de reglas fue originalmente pensado para ayudar a contrarrestar las afirmaciones de que eXtreme Programming falla en apoyar algunas de las disciplinas prominentes necesarias para el desarrollo moderno.

Planificación

- Se escriben historias de usuario.
- Los planes de entrega definen el calendario de entregas.
- Realizar pequeños, pero frecuentes, entregables.
- El proyecto se divide en iteraciones.
- Cada iteración comienza con su planificación.

Gestión

- Proporcionar al equipo un espacio de trabajo abierto y dedicado.
- Establecer un ritmo sostenible.
- Una reunión de pie se inicia cada día.
- Se mide la velocidad del proyecto.
- Mover a la gente.
- Reparar eXtreme Programming cuanto éste no funciona.

Diseño

- Simplicidad
- Escoger un sistema de metáforas.

- Utilizar cartas CRC [2] para las sesiones de diseño.
- Investigar para reducir el riesgo.
- Evitar añadir funcionalidad en etapas tempranas.
- Refactorizar siempre que sea posible todo lo que sea posible.

Codificación

- El cliente deber estar siempre disponible.
- El código debe escribirse siguiendo estándares.
- Crear primero una prueba unitaria.
- Todo el código de producción se realiza por parejas.
- Solo una pareja realiza la integración de su código a la vez.
- Realizar pruebas de integración.
- Dedicar una máquina para realizar pruebas de integración.

Pruebas

- Todo el código debe tener una prueba unitaria.
- Todo el código debe pasar todas las pruebas unitarias antes de poderse desplegar.
- Si se detecta un fallo se debe crear una prueba unitaria.
- Se deben realizar pruebas de aceptación con frecuencia y se deben publicar los resultados.

8.3.3 Prácticas Extremas

Basadas en lo que se consideraba por aquél entonces las "mejores prácticas de desarrollo de software", estas doce Mejores Prácticas de eXtreme Programming detallan los procedimientos específicos que deben seguirse al implementar un proyecto usando eXtreme Programming.

8.3.3.1 Retroalimentación Granular

*Programación por parejas**

En esencia, la programación por parejas significa que dos personas trabajan en el mismo sistema al desarrollar cualquier código de producción. Al rotar los miembros de del equipo con frecuencia, eXtreme Programming promueve una mejor comunicación y la creación de equipos.

Planificaicón de Entregables

[2] Las tarjetas CRC (Class-responsibility-collaboration) son una herramienta brainstorming utilizadas en el diseño de software orientado a objetos.

A menudo esto toma la forma de una reunión en un intervalo bien definido (cada una o dos semanas), y es donde la mayoría de la planificación para el proyecto tienen lugar.

Dentro de este procedimiento existe la etapa de "Planificación de Despliegue", donde se discute y toman decisiones sobre lo necesario para desplegar entregas inminentes. Esta planificación incluye:

- Fase de exploración: Se utilizan historias de usuario para determinar aquellos requisitos más importantes para el cliente.
- Fase de compromiso: Se establecen planificaciones y compromisos de equipo para cumplir con el siguiente plan de entrega.
- Fase de reajuste: Esta fase permite ajustar planes anteriores en función de las circunstancias actuales del proyecto, similar a muchas otras metodologías ágiles de desarrollo.

A continuación, comienza la planificación de iteraciones, que consiste en las mismas tres fases anteriores, pero con variantes en su implementación:

- Fase de exploración: Se escriben todas los requisitos del proyecto.
- Fase de compromiso: Se asignan y planifican todas las actividades y tareas sin terminar necesarias para cumplir con la siguiente iteración.
- Fase de reajuste: Aquí es donde el desarrollo ocurre y, una vez terminado, la iteración resultante se compara con las historias de usuario creadas al inicio del proceso de planificación.

Desarrollo Guiado por Pruebas

Aunque se podría dedicar todo un artículo sobre qué es TDD, el concepto es bastante bien conocido entre desarrolladores y en esencia significa que se generan pruebas para cada requisito del proyecto, y una vez generadas es cuando se desarrolla el código que debe pasar por estas pruebas.

El equipo completo

Como ocurre en muchos otros métodos SDLC y prácticas, eXtreme Programming promueve la integración del cliente durante todo el proceso, utilizando sus sugerencias y comentarios para ayudar a moldear el proyecto durante todas sus etapas.

8.3.3.2 Proceso Continuo

Integración continua

Otra práctica común en el desarrollo moderno, la idea detrás de la integración continua es que todo el código desarrollado por el equipo se combina en un repositorio común muchas veces al día. Esto asegura que los problemas con la integración en todo el proyecto se descubran y se resuelvan lo antes posible.

Refactorización de código

Otra práctica muy común, la idea detrás de la refactorización de código es simplemente mejorar y rediseñar la estructura del código ya existente, sin modificar su comportamiento fundamental.

Ejemplos simples de refactorización incluyen tareas como renombrar variables por otros más descriptivos o reducir el código repetido encapsulándolo en métodos o funciones reutilizables.

Entregables pequeños

Similar al Modelo Iterativo, este concepto asegura que el proyecto contará con pequeñas emisiones de forma frecuente, permitiendo al cliente, así como a todos los miembros del equipo, tener una idea de cómo se está desarrollando el proyecto.

8.3.3.3 Comprensión

Estándares de codificación

El estándar de codificación es simplemente un conjunto de mejores prácticas dentro del propio código (como el formato y el estilo) que todo el equipo cumple a lo largo de todo el ciclo de vida del proyecto. Esto promueve una mejor comprensión y legibilidad del código no sólo para los miembros actuales, sino también para los futuros desarrolladores.

Propiedad compartida del código

Esta práctica permite a cualquier desarrollador del equipo cambiar cualquier sección del código, según sea necesario. Aunque esta práctica puede parecer peligrosa para algunos, acelera el tiempo de desarrollo, y cualquier problema potencial puede ser solventado con pruebas de unitarias.

Simplicidad

No hay razón para complicar las cosas siempre que exista una opción más sencilla. Esta práctica intenta mantener todos los componentes y el código tan simple como sea posible, de este modo el equipo está siempre evaluando si las cosas se podrían hacer de una manera más fácil.

Sistema de metáforas

Generalmente visto como parte de los estándares de codificación, el sistema de metáforas trata sobre que cada persona en el equipo debe ser capaz de mirar el código de alto nivel que se desarrolla, y tener una comprensión clara sobre qué hace el código.

8.3.3.4 Bienestar del Programador

Ritmo sostenible

Un concepto clave para un mejor equilibrio entre el trabajo y la vida privada de los desarrolladores de un proyecto eXtreme Programming es la noción de que nadie debería estar obligado a trabajar en exceso en una semana de trabajo. Las horas extraordinarias son mal vistas, al igual que el concepto de “crunch time”, donde se espera que los desarrolladores trabajen una cantidad de horas elevada para cumplir con un plan de entrega.

8.4 CakePHP Framework

CakePHP es un framework web de código abierto. Sigue el paradigma *MVC* y está escrito en *PHP*, basado en los conceptos de Ruby on Rails y distribuido bajo la Licencia MIT. Utiliza conceptos de ingeniería de software bien conocidos y patrones de diseño de software, tales como convención sobre configuración, controlador de vista de modelo, registro activo, asignación de datos de asociación y controlador frontal.

CakePHP nace en abril de 2005, cuando un programador polaco llamado Michal Tatarynowicz escribió una versión simplificada de un framework PHP. Publicó este framework bajo la licencia MIT, y lo abrió a la comunidad de desarrolladores. En diciembre de 2005, L. Masters y G. J. Woodworth fundaron Cake Software Foundation para promover el desarrollo relacionado con CakePHP. La versión 1.0 fue lanzada en mayo de 2006.

8.4.1 Paradigma MVC y CakePHP

La aplicación *CRM* de *BDA* sigue el paradigma *MVC* como base para su arquitectura, paradigma que es en realidad definido por el framework base, *CakePHP*, sobre el cual están construidos todos los componentes software. Y que como bien su nombre indica, *MVC* (Model-View-Controller), divide la lógica de la aplicación en tres partes:

- **Model**: El modelo representa la parte de la aplicación que implementa la lógica de negocio. Esto significa que es responsable de la recuperación de datos convirtiéndolos en conceptos significativos para la aplicación, así como su procesamiento, validación, asociación y cualquier otra tarea relativa a la manipulación de dichos datos.
- **View**: La vista hace una presentación de los datos del modelo estando separada de los objetos del modelo. Es responsable del uso de la información de la cual dispone para producir cualquier interfaz de presentación de cualquier petición que se presente.
- **Controller**: La capa del controlador gestiona las peticiones de los usuarios. Es responsable de responder la información solicitada con la ayuda tanto del modelo como de la vista. Los controladores pueden ser vistos como administradores cuidando de que todos los recursos necesarios para completar una tarea se deleguen a los trabajadores más adecuados. Espera peticiones de los clientes, comprueba su validez de acuerdo a las normas de autenticación o autorización, delega la búsqueda de datos al modelo y selecciona el tipo de respuesta más adecuado según las preferencias del cliente. Finalmente delega este proceso de presentación a la capa de la Vista.

8.4.2 Flujo de una Petición Web

El flujo de una petición típica comienza cuando un usuario solicita una página o un recurso. A un nivel muy alto de abstracción esta petición pasa por las siguientes etapas para ser atendida:

1. La petición es inicialmente procesada por distintos enrutadores (Routers).

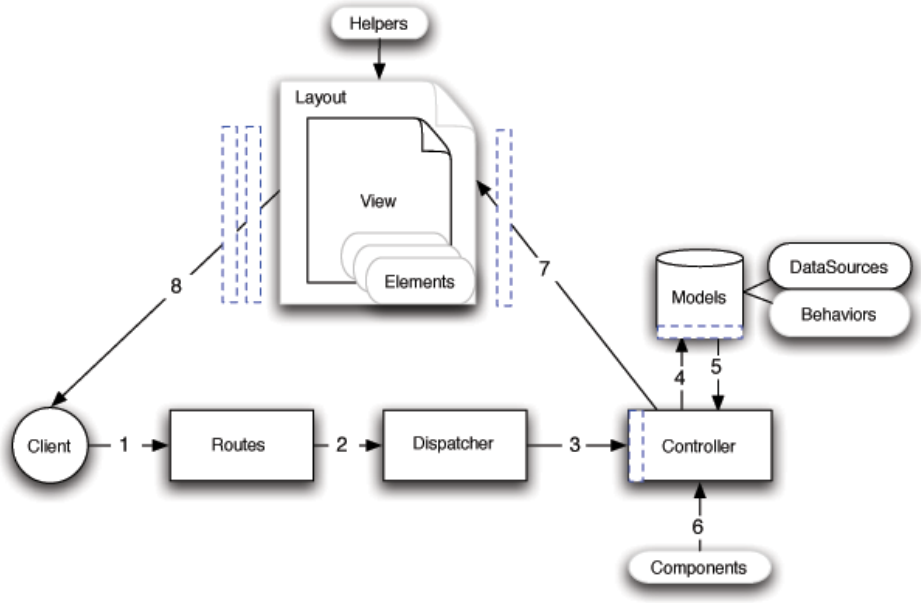


Figura 8.1: Esquematización de una petición *MVC* en CakePHP.
Muestra un ejemplo sencillo de una petición *MVC* en CakePHP. A efectos ilustrativos, se puede suponer que un usuario ha realizado clic sobre un enlace de alguna de las páginas de la aplicación.

2. La petición, después de haber sido correctamente enrutada, el despachador (Dispatcher) busca e inicializa el controlador (clase PHP) adecuado para atender la petición.
3. Se invoca la acción del controlador correspondiente (método del objeto anterior) para atender la petición, el controlador interactúa con los modelos y componentes correspondientes en caso de ser necesario.
4. El controlador delega en la vista la tarea de generar la presentación resultante de los datos proporcionada por el modelo.
5. Finalmente, cuando esta presentación se genera, se envía de inmediato al usuario.

8.5 Tolerancia Ante Fallos

La tolerancia ante fallos es una propiedad de un sistema que le permite seguir funcionando ante un posible fallo (uno o varios) de alguno de sus componentes. Un sistema tolerante a fallos es aquél que continúa trabajando correctamente cuando uno se presentan fallos en alguno de sus elementos, tanto hardware como software. El concepto de tolerancia a fallos puede aumentar la fiabilidad, pero ello no presupone que el sistema tolerante a fallos tenga una alta fiabilidad. *[AntonioCreusSole2005]*

Si la calidad operacional del sistema decrece con el tiempo, entonces este decrecimiento resulta proporcional a la gravedad del posible fallo, en contraste con un sistema diseñado de forma inapropiada en donde un fallo minúsculo puede ocasionar una caída completa del sistema. La tolerancia

a fallos es necesaria en sistemas altamente disponibles o críticos. La capacidad de un sistema para mantener sus funcionalidades aun cuando partes del sismo dejan de hacerlo correctamente se denomina degradación graduada. *[OscarGonzales1997]*

Existen múltiples estrategias para conseguir un sistema tolerante a fallos. Las más importantes son la redundancia, la replicación y la autocorreción.

- **Redundancia**: La redundancia es la técnica de tolerancia a fallos que consiste en añadir más medios de información, fuentes o tiempo del que es necesario para la operación normal del sistema. Existen varias formas de redundancia:
  - Uso de códigos detectores y correctores de error.
  - Tener módulos pasivos que hacen exactamente lo mismo que otros activos de forma que puedan sustituirlo y evitar que sistema se caiga.
  - Redundancia modular. Consiste en tener un número normalmente impar (para evitar luego empates) de módulos que hacen la misma función, aunque pueden implementarla de forma diferente. Luego hay un módulo (el cual puede tener a su vez redundancia modular) que evalúa las salidas de dichos módulos y toma como resultado global el resultado que devuelve la mayoría de los módulos redundantes.
- **Replicación**: Para evitar que un fallo produzca la pérdida de la información almacenada en un sistema se suele replicar esa información en más de un soporte físico (redundancia), o en un equipo o dispositivo externo a modo de respaldo. De esta forma, si se produce alguna falla que pueda ocasionar pérdida de datos, el sistema debe ser capaz de restablecer toda la información, recuperando los datos necesarios a partir de algún medio de respaldo disponible. En esto se basa el sistema de almacenamiento en RAID (Redundant Array of Independent Disks). Los sistemas RAID (a excepción de RAID 0) se basan en la técnica mirroring ("espejo"), que permite la escritura simultánea de los datos en más de un disco del array. En sistemas distribuidos es frecuente replicar la información para conseguir que sean tolerantes a los fallos. Para hacer que dicha información sea consistente en todo el sistema distribuido se implementan protocolos de consenso.
- **Autocorrección**: Esta estrategia es la que hacen los navegadores de Internet. Cuando el navegador de Internet envía una solicitud HTTP al servidor web este responde con el contenido del sitio en formato estandarizado HTML o XHTML, si este código viene con errores (el estándar no se cumple), entonces el navegador es libre de elegir qué hacer con él, ya sea no mostrar el contenido con problemas, intentar corregirlo o simplemente mostrarlo en texto plano. Normalmente lo que hacen es intentar corregirlo.

8.6 Tecnologías de Virtualización y sus Beneficios

Las tecnologías de virtualización son relativamente nuevas en el mundo IT (10 años aprox.), ofrecen un desempeño, flexibilidad y posibilidades que resultan simplemente inalcanzables en un entorno físico o "tangible". Y aunque a día de hoy estas tecnologías continúan madurando y evolucio-

nando, muchas organizaciones no aprovechan al completo el potencial que este tipo de tecnologías son capaces de ofrecer; se estancan por lo general en entornos donde la virtualización se aleja bastante del total que es posible virtualizar.

Los beneficios que este tipo de tecnologías pueden aportar a organizaciones IT son múltiples. Algunos beneficios pueden ser, por ejemplo:

Ahorro energético: Migrar servidores físicos a versiones virtualizadas de los mismos, a la vez que reduce hardware requerido, reduce además los costes derivados del consumo eléctrico y de los sistemas de enfriamiento necesarios.

Laboratorios/entornos QA: La virtualización permite desplegar de forma rápida y sencilla entornos auto-contenidos de prueba para el aseguramiento de la calidad.

Reducción de dependencias de hardware: Aunque no siempre resulta ser una desventaja, en ocasiones depender demasiado de determinados fabricantes de hardware o de modelos concretos de éstos, puede en algunos casos resultar extremadamente problemático. La virtualización se abstrae por completo del hardware base y lo reemplaza por una versión virtual del mismo.

Aislamientos de aplicaciones: En entornos físicos, por lo general los centros de datos emplean la filosofía de "una aplicación, un servidor/máquina" para conseguir un aislamiento efectivo. Sin embargo, esto produce un incremento muchas veces innecesario del número de servidores físicos necesarios, aumentando así los costes y el número de recursos infrautilizados. La virtualización de servidores proporciona un aislamiento de aplicaciones al consolidar muchas de estas máquinas virtuales sobre un número reducido de máquinas físicas.

Mejorar la recuperación ante desastres: La virtualización ofrece tres importantes componentes a la hora de idear un plan de recuperación ante desastres.

1. Capacidad de abstraerse del hardware. Al eliminar las dependencias de hardware, no es necesario mantener copias idénticas del hardware de producción en caso de ser necesario su reemplazo.
2. Al consolidad los servidores en un conjunto reducido de máquinas físicas, es mucho más simple, y económico, crear un sitio de replicado.
3. La mayoría de software orientado a virtualización ofrecen mecanismos que permiten automatizar las tareas de conmutación ante fallos en caso de ser necesario desplegar un plan de recuperación.

8.6.1 Proxmox Virtual Environment

ProxmoxVE es un proyecto de código abierto, mantenido y desarrollado por *Proxmox Server Solutions* junto con el apoyo financiero de *Internet Foundation Austria*. Se trata de una plataforma completa de virtualización basada en sistemas de código abierto que permite la virtualización ba-

sada tanto en LXC [2] como KVM [3].

Es una distribución bare-metal, basada en *Debian* con solo los servicios básicos para de esta forma obtener un mejor rendimiento. ProxmoxVE, no es solo una máquina virtual más, con una interfaz gráfica muy sencilla esta herramienta permite la migración en vivo de máquinas virtuales, clustering de servidores, backups automáticos y conexión a un NAS/SAN con NFS [13], iSCSI, etc. *[EliasHidalgo2012]*

Al utilizar LXC se puede cambiar tanto memoria RAM como espacio en disco asignados, en tiempo real y sin reiniciar el sistema. Otra cosa muy interesante son las plantillas, que consisten en un sistema operativo con algún software pre-instalado, que se descargan directamente desde la interfaz de administración y que permiten crear una máquina virtual a partir de ellas. Las principales características de ProxmoxVE son:

- Es de código abierto
- Permite la migración en vivo
- Dispone de una alta habilitación de puentes de red
- Plantillas de construcción de S.O.
- Copias de seguridad programadas
- Herramientas de línea de comandos

8.6.1.1 Modelos de Almacenamiento

ProxmoxVE soporta almacenamiento local con grupos LVM [4], directorios y ZFS [5], así como almacenamiento basados en red de como iSCSI [6], Fibre Channel [7], NFS [13], GlusterFS [9], CEPH [10] o DRBD [11].

[2] LXC (Linux Containers) es una tecnología de virtualización en el nivel de sistema operativo (SO) para Linux.

[3] Kernel-based Virtual Machine o KVM, (en español, Máquina virtual basada en el núcleo) es una solución para implementar virtualización completa con Linux.

[13] El Network File System, o NFS, es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuido en un entorno de red de computadoras de área local.

[4] LVM es una implementación de un administrador de volúmenes lógicos para el kernel Linux.

[5] ZFS es un sistema de archivos y volúmenes desarrollado por Sun Microsystems para su sistema operativo Solaris.

[6] iSCSI (Abreviatura de Internet SCSI) es un estándar que permite el uso del protocolo SCSI sobre redes TCP/IP.

[7] El canal de fibra (del inglés fibre channel) es una tecnología de red utilizada principalmente para redes de almacenamiento.

[9] El Sistema de Archivos Gluster, Gluster File System o GlusterFS, es un sistema de archivos para NAS desarrollado inicialmente por *Gluster Inc*.

[10] Ceph File System es un sistema de archivos distribuido libre, está diseñado para el uso con gran cantidad de datos, está muy enfocado para el uso con Big Data.

[11] Un sistema de replicación para almacenamiento distribuido para Linux. Es utilizado generalmente en clústeres de alta disponibilidad.

8.6.1.2 Cluster de Alta Disponibilidad

ProxmoxVE puede ser agrupado a través de múltiples nodos de servidor. Desde la versión 2.0, ProxmoxVE ofrece una opción de alta disponibilidad para grupos basados en la pila de comunicaciones *Corosync* [12]. Los servidores virtuales individuales pueden ser configurados para su alta disponibilidad, utilizando la suite *Red Hat cluster*. Si un nodo no está disponible o bien un servidor virtual falla éste puede ser movido de forma automática a otro nodo y reiniciado.

8.6.1.3 Migración en Vivo

En un cluster de alta disponibilidad, las máquinas virtuales en ejecución pueden ser trasladadas de un nodo físico a otro sin interrupciones en los servicios. Esto resulta de utilidad, por ejemplo, en situaciones de mantenimiento de uno de los nodos físicos o cuando se desea simplemente liberar la carga de uno de ellos.

8.6.2 Otras Alternativas

Antes de adoptar la decisión final de utilizar ProxmoxVE, distintas alternativas fueron analizadas y comparadas. A continuación, se ofrece una descripción de las tres alternativas más destacadas durante este proceso y una tabla resumen de las características de cada una de estas herramientas. Para mayor información sobre cada producto, se invita al lector a consultar los hipervínculos al pie de página.

ProxmoxVE fue escogido por resultar mucho más simple de gestionar al contar con un centro de gestión unificado, por resultar económicamente atractivo (gratuito con suscripción opcional) y principalmente por estar enfocado en **virtualización eficiente de entornos Linux** con LXC; la totalidad de los servicios ofrecidos por la aplicación *CRM* de *BDA* están diseñados funcionar sobre entornos Linux.

- **VMware vSphere**: VMware vSphere es la plataforma de virtualización líder del sector para construir infraestructuras de cloud. Permite a los usuarios ejecutar aplicaciones críticas para el negocio con confianza y responder con mayor rapidez a las necesidades empresariales. vSphere acelera el cambio hacia el cloud computing para los centros de datos existentes, además de sustentar las ofertas de cloud pública, de tal forma que constituye la base para el único modelo de cloud híbrida del sector. Con más de 250 000 clientes en todo el mundo y la compatibilidad con más de 2500 aplicaciones de más de 1400 partners ISV, VMware vSphere es la plataforma de confianza para cualquier aplicación. [15].
- **Windows Hyper-V**: Microsoft Hyper-V es un programa de virtualización basado en un hipervisor para los sistemas de 64-bits con los procesadores basados en AMD-V o Tecnología de virtualización Intel (el instrumental de gestión también se puede instalar en sistemas x86).

[12] Un un grupo de sistemas de comunicación con funciones adicionales para implementar aplicaciones de alta disponibilidad.

[15] https://www.vmware.com/files/es/pdf/VMware-vSphere-Entreprise-Edition-Datasheet.pdf

Una versión beta de Hyper-V se incluyó en el Windows Server 2008 y la versión definitiva se publicó el 26 de junio de 2008. [16].

- **Cetrix XenServer**: Citrix XenServer es una plataforma de virtualización de servidores administrada, completa e integrada en el potente hipervisor Xen. La tecnología Xen es reconocida como el software de virtualización más rápido y seguro de la industria. XenServer está diseñado para una gestión eficiente de los servidores virtuales de Windows® y Linux® y proporciona una consolidación rentable de los servidores y la continuidad del negocio. [14].

[16] https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/about/
[14] http://www.spetel.com/portfolio/citrix-xenserver-2/

Característica	Proxmox VE	VMware vSphere	Windows Hyper-V	Citrix XenServer
SO invitados admitidos	Windows y Linux (KVM). Otros sistemas son soportados y mantenidos por la comunidad.	Windows, Linux, UNIX	Versiones modernas de Windows, soporte para Linux limitado.	Mayoría de versiones de Windows, soporte para Linux limitado.
Open Source	si	no	no	si
Linux Containers (LXC)	si	no	no	no
Gestión unificada	si	Requiere de un servidor dedicado para la gestión.	Requiere de un servidor dedicado para la gestión.	si
Alta disponibilidad	si	si	Require un cluster "Microsoft Failover", limitado a ciertos SO invitados	si
Instantáneas en tiempo real: Backup en caliente de una VM	si	si	limitado	si
Bare metal hypervisor	si	si	si	si
Migración en caliente de VM	si	si	si	si
RAM y CPU máx. por anfitrión	160 CPU/2 TB Ram	160 CPU/2 TB Ram	64 CPU/1 TB Ram	se desconoce

Tabla 8.1: Comparativa, distintas soluciones de virtualización

CAPÍTULO 9

Anexos

A continuación se indican todos los documentos adjuntos; una breve descripción e información relevante para su interpretación en caso de resultar necesario. Todo el material en cuestión se haya al final de esta memoria, después del separador identificado como “ANEXOS”.

9.1 Diagrama de Datos EAV

Representación gráfica de los distintos propiedades virtuales y no virtuales, última revisión septiembre 2017. Cómo interpretar el diagrama:

- Todos los elementos rectangulares representan tablas físicas o atributos no virtuales.
- Todos los elementos ovales representan elementos virtuales. - El óvalo central representa un bundle. - Los óvalos más pequeños asociados a un bundle representan propiedades virtuales.
- **Cada tipo de dato se representa por un color.**
    - Verde: Números enteros.
    - Rojo: Número de coma flotante.
    - Azul: Cadenas de texto.
    - Ámbar: Valores booleanos.
    - Púrpura: Marcas temporales, fechas u horas.

Por motivos de legibilidad en su versión impresa, el diagrama se segmentado en varios cuadrantes o vistas detalle. Sin embargo, para una mejor visualización se recomienda la versión digital de este documento.

9.2 Bundles EAV

A continuación, en la tabla siguiente se ofrece un listado completo de atributos virtuales utilizados en el *CRM* de *BDA*; se indican “bundle”, nombre, tipo de dato y visibilidad.

9.3 EAV Plugin

Véase el documento adjunto titulado como “EAV Plugin”; un extracto de la documentación en línea que puede ser consultada en su totalidad en QuickBook [1].

[1] http://book.quickappscms.org/developers/eav-api.html#eav-api

9.4 Modelo Relacional

Diagrama de despliegue, última revisión septiembre de 2016.

9.5 Diagrama de Red

Diagrama de despliegue, última revisión agosto de 2017.

9.6 Diagrama de Despliegue

Diagrama de despliegue, última revisión julio de 2017.

9.7 Relación Componentes Físicos y Virtuales

Disposición de servidores virtuales dentro de cluster de virtualización ProxmoxVE. Última revisión, julio 2017.

9.8 Relación Hardware y Software

Cada componente software es desplegado y utilizada con combinación con uno o más componentes de tipo hardware. La tabla de a continuación describe el hardware, sistemas operativos y software/servicios utilizados en cada componente hardware.

9.9 Template Historias de Usuario

Plantilla (template) de GitHub orientado a la toma de requisitos. En formato *Markdown*.

9.10 Template Recogida de Incidencias

Plantilla (template) de GitHub orientado al registro de incidencias/bugs. En formato *Markdown*.

9.11 Backlog

Backlog de historias de usuario, limitado a últimos cien (100) del mes de agosto de 2017.

9.12 Visión Proyecto CRM

Gráficos de situación del repositorio; proyecto *CRM* de *BDA*.

9.13 Grafo Ramas GIT

Gráfico disposición y evolución de ramas GIT. Repositorio de código fuente proyecto *CRM* de *BDA*. Captura limitada al mes de agosto de 2017. Por motivos de privacidad alguna información puede haber sido censurada.

9.14 Minería de Procesos

Véase el documento adjunto titulado como “Minería de Procesos”.

Índice de figuras

Índice de cuadros

Bibliografía

[DomingoRey2012] Domingo Rey Peteiro. Sinapsys Business Solutions, S.L. La Gestión Tradicional y la Gestión por Procesos

[KlintFinley2012] Klint Finley. “What Exactly Is GitHub Anyway?”, techcrunch.com.

[AlejandroBedini2013] Alejandro Bedini González, “Gestión de Proyectos de Software”

[ZhenyuFang2010] Zhenyu Fang, Changqing Yin. BPM Architecture Design Based on Cloud Computing.

[KenBeck1999] Extreme Programming Explained: Embrace Change. Kent Beck, 1999.

[AntonioCreusSole2005] Fiabilidad y seguridad: su aplicación en procesos industriales. 2005, p193.

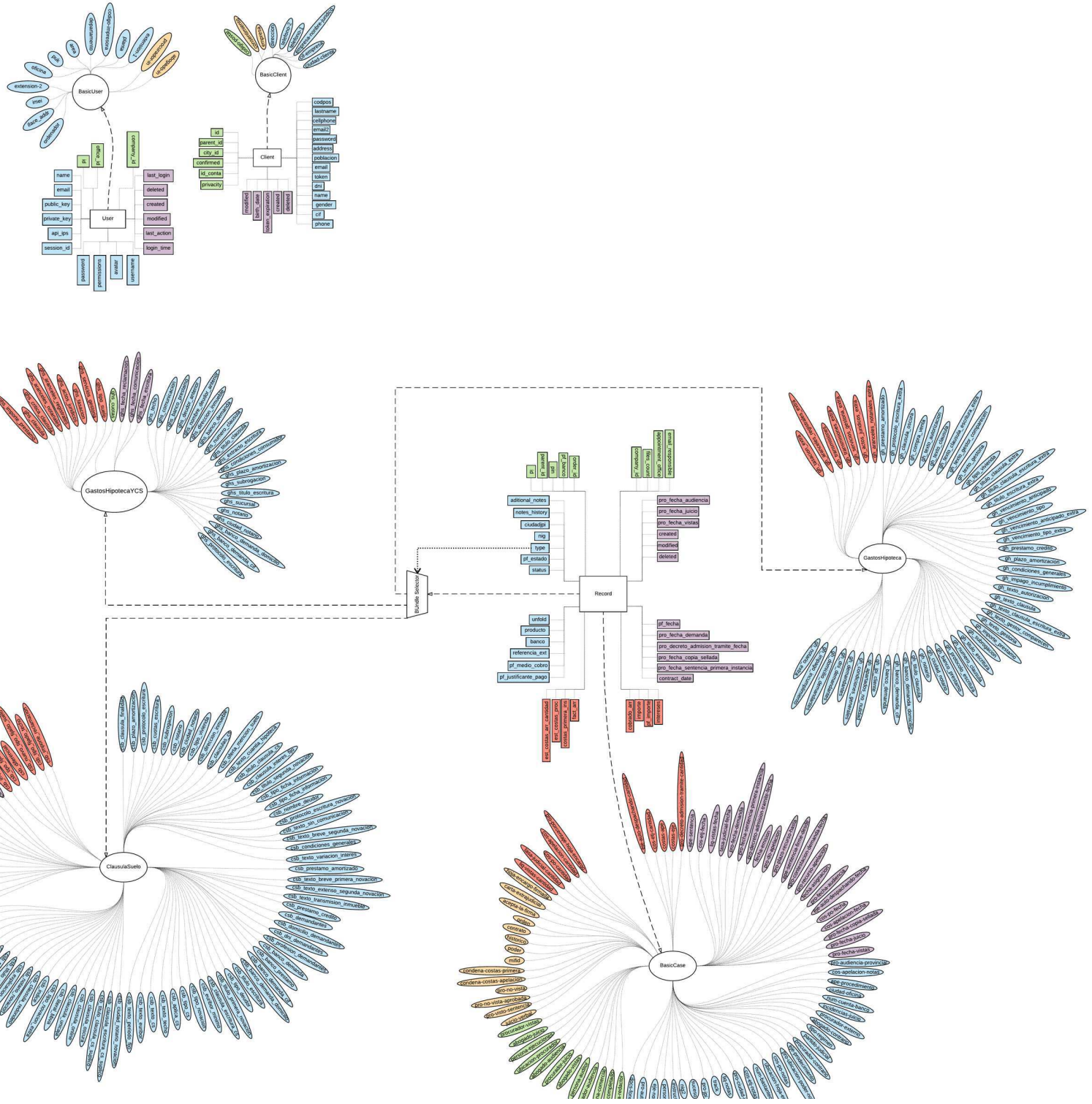
[OscarGonzales1997] Adaptive Fault Tolerance and Graceful Degradation, Oscar González et al., 1997, University of Massachusetts - Amherst

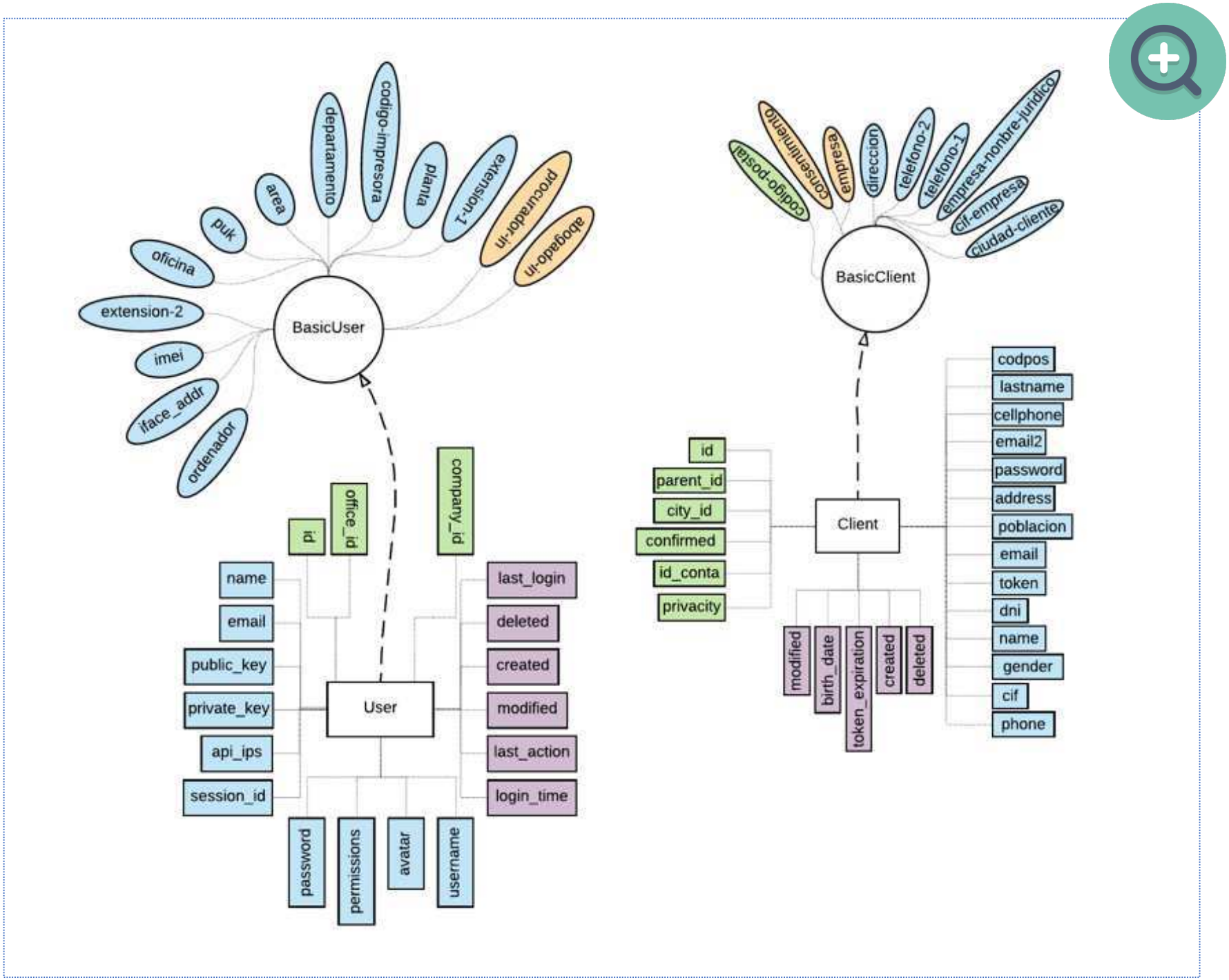
[EliasHidalgo2012] Elias Hidalgo. Proxmox VE, una gran herramienta de virtualización. 2012, http://linuxzone.es.

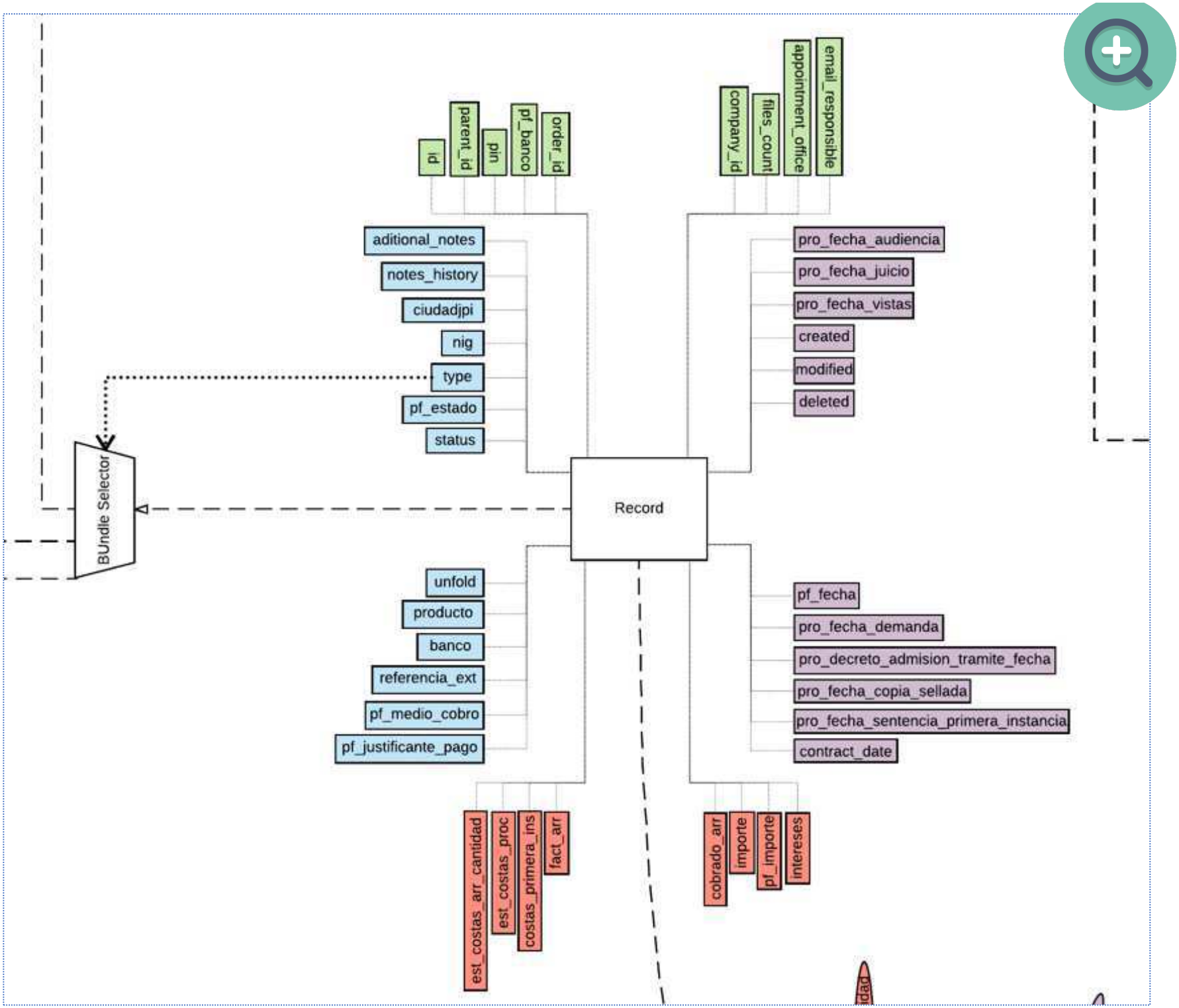
ANEXOS

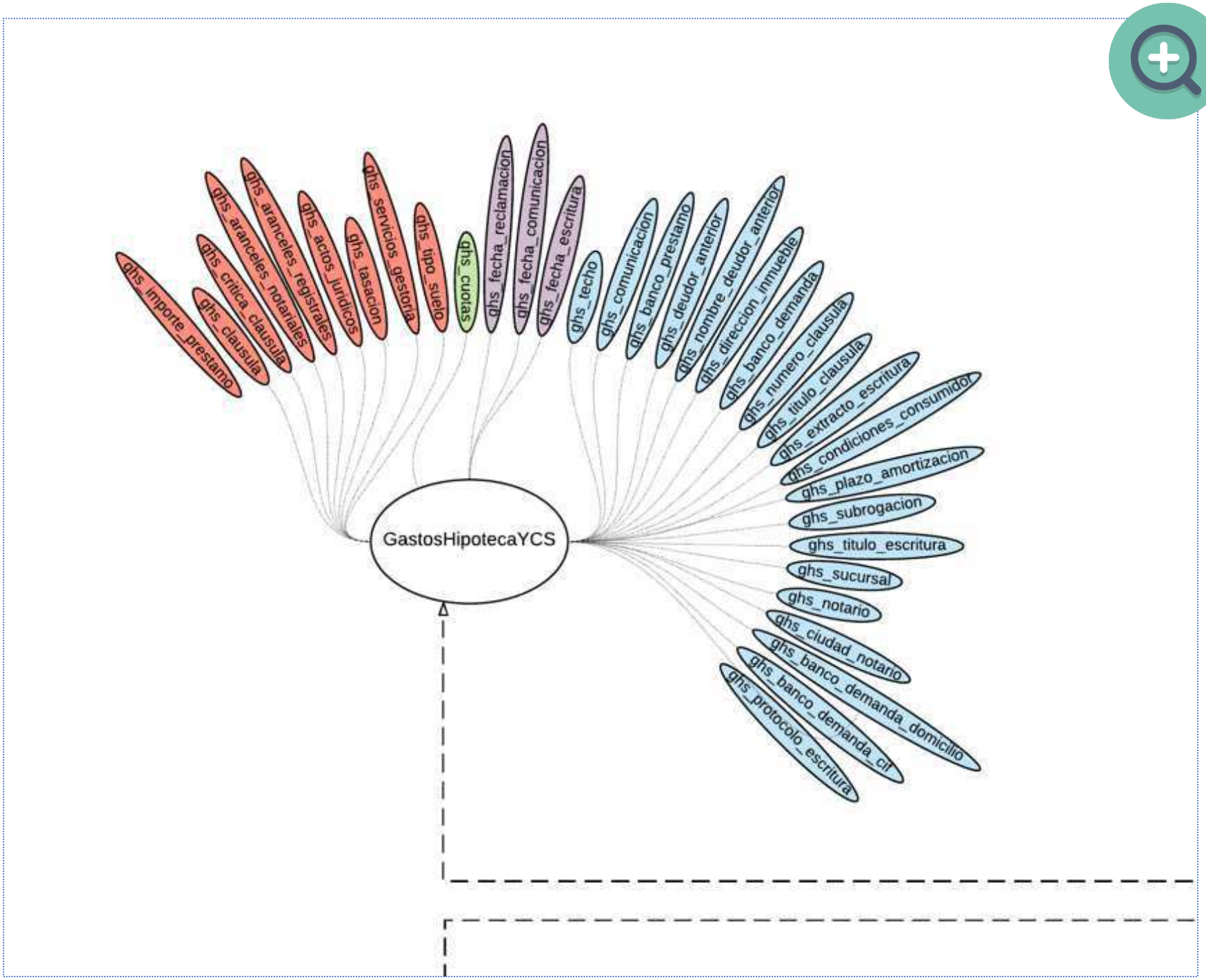
ANEXO

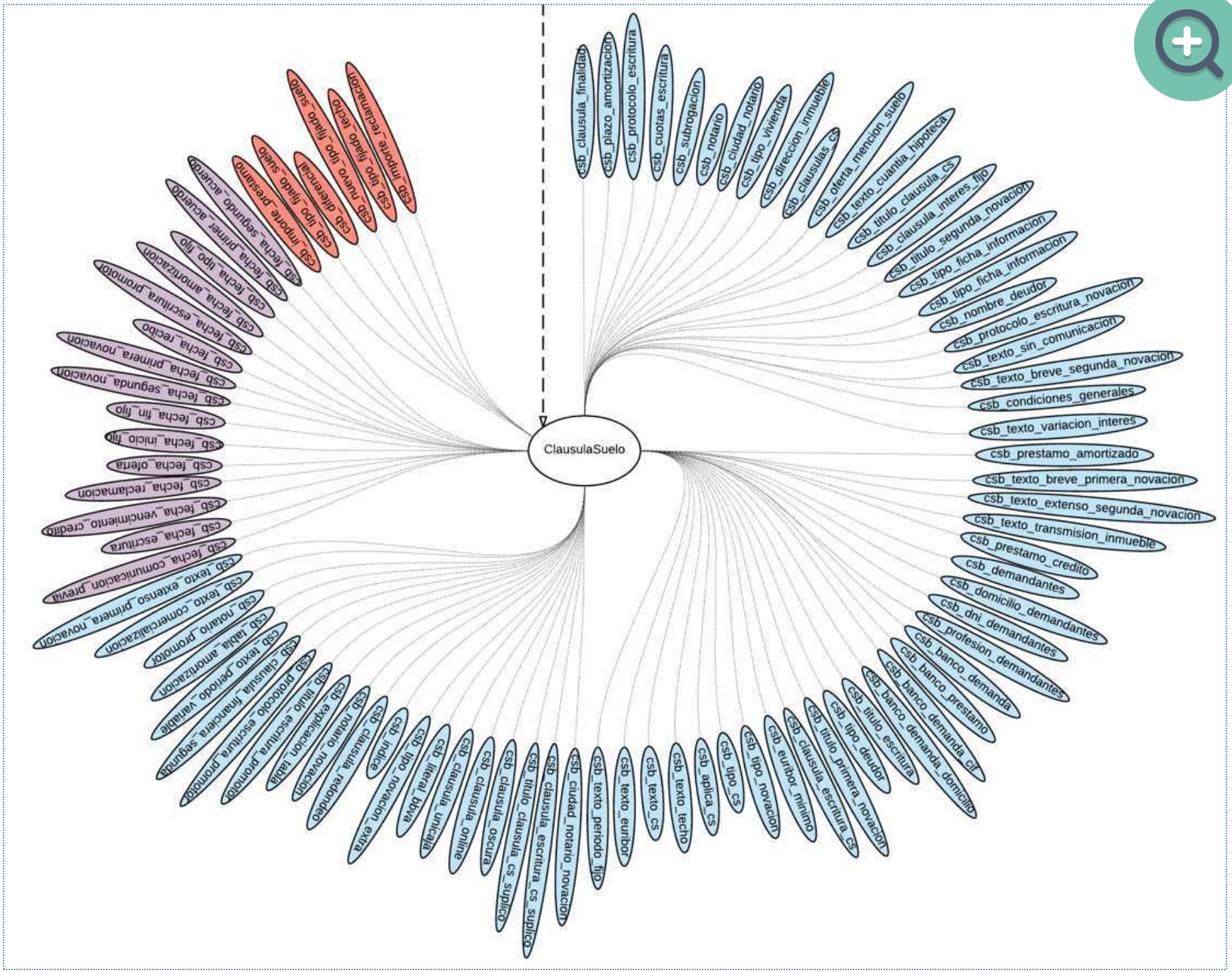
Diagrama de Datos EAV

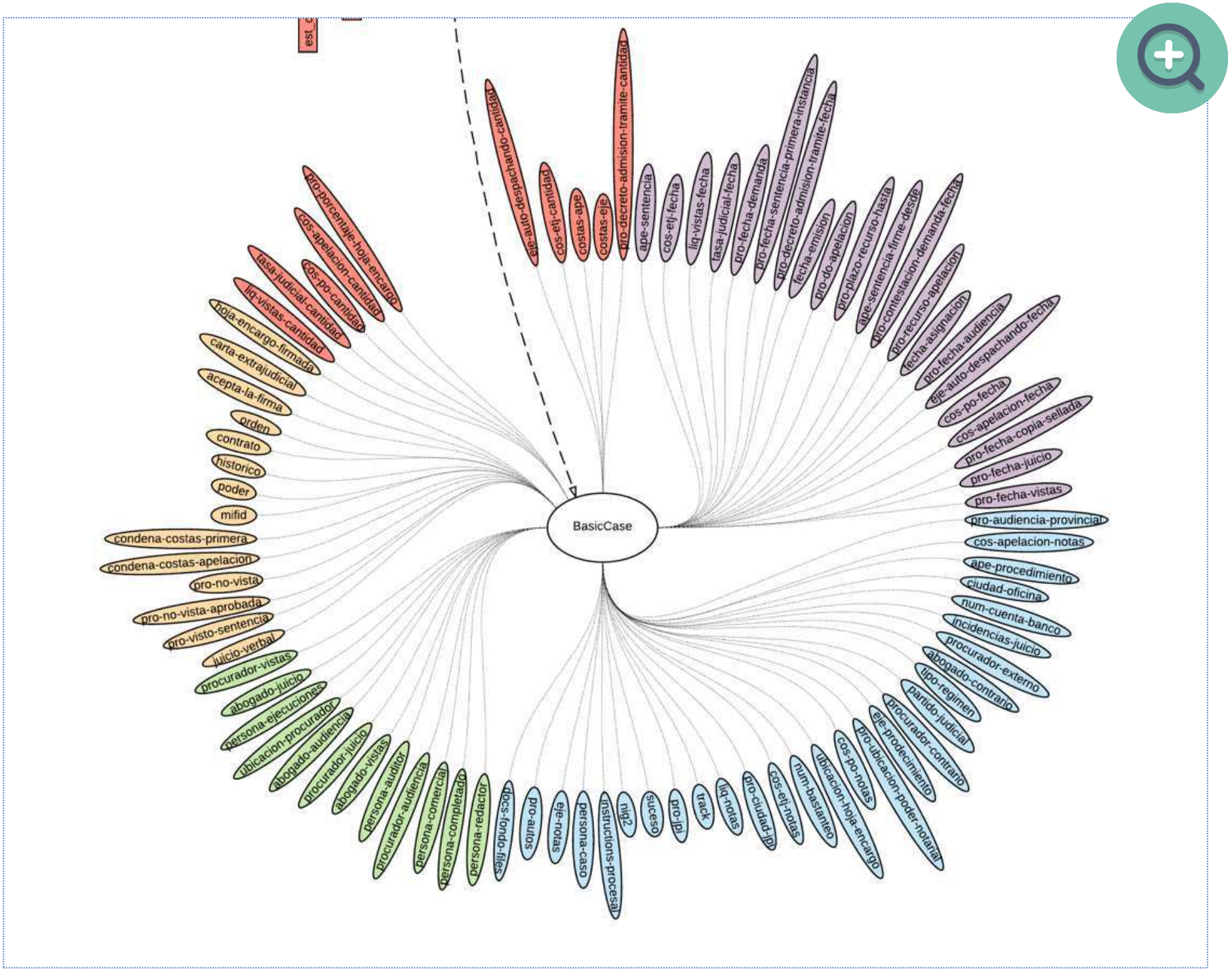

BasicUser
BasicClient
Client
User
GastosHipotecaYCS
GastosHipoteca
Record
Bundle Selector
ClausulaSuelo
BasicCase

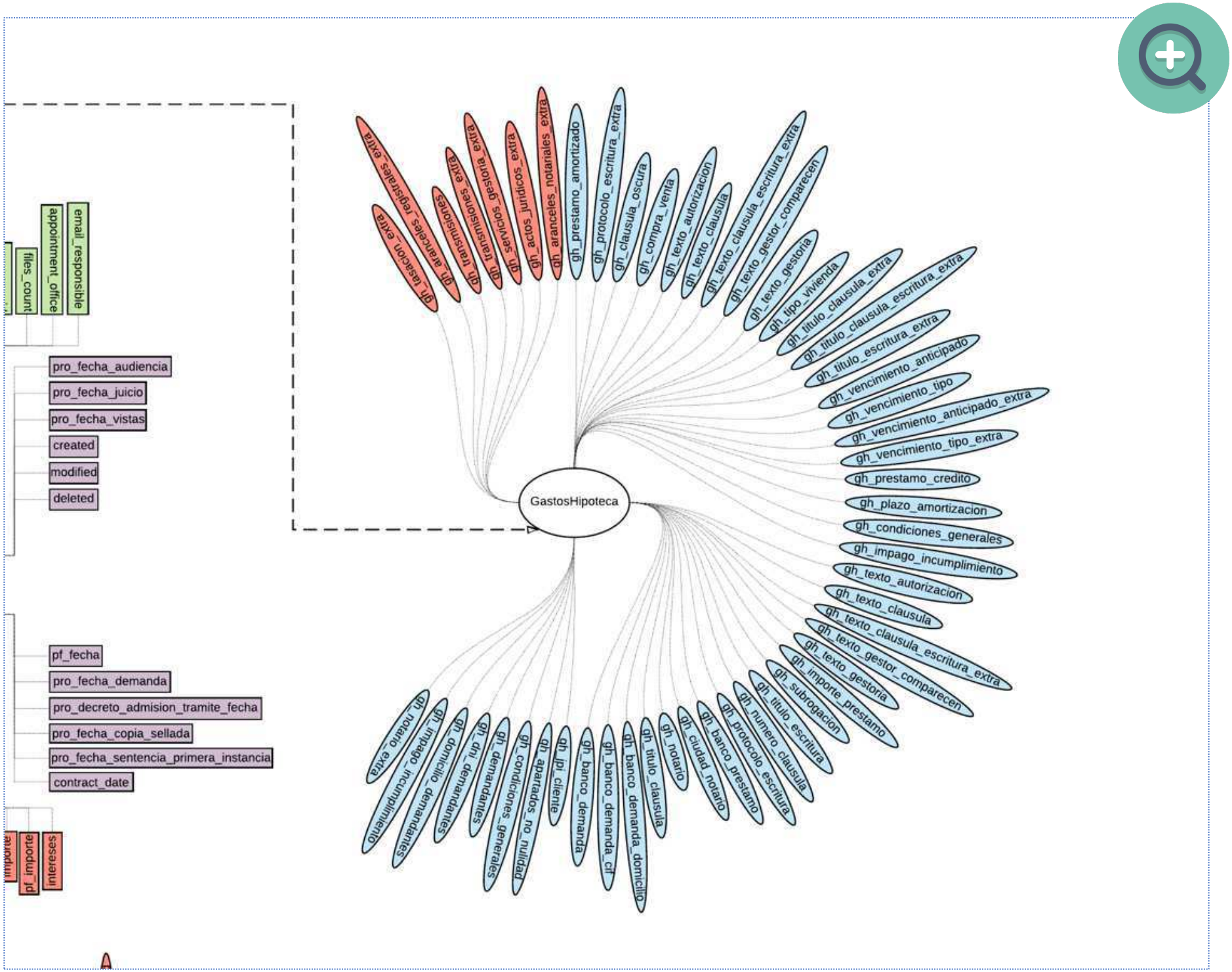

BasicUser
codigo-impresora
departamento
area
planta
extension-1
procurador-ln
abogado-ln
puk
oficina
extension-2
imei
iface_addr
ordenador
User
id
office_id
company_id
name
email
public_key
private_key
api_ips
session_id
last_login
deleted
created
modified
last_action
login_time
password
permissions
avatar
username
BasicClient
codigo-postal
consentimiento
empresa
direccion
telefono-2
telefono-1
empresa-nombre-juridico
cif-empresa
ciudad-cliente
Client
id
parent_id
city_id
confirmed
id_conta
privacity
codpos
lastname
cellphone
email2
password
address
poblacion
email
token
dni
name
gender
cif
phone
modified
birth_date
token_expiration
created
deleted


id
parent_id
pin
pf_banco
order_id
company_id
files_count
appointment_office
email_responsible
aditional_notes
notes_history
ciudadjpi
nig
type
pf_estado
status
pro_fecha_audiencia
pro_fecha_juicio
pro_fecha_vistas
created
modified
deleted
BUndle Selector
Record
unfold
producto
banco
referencia_ext
pf_medio_cobro
pf_justificante_pago
pf_fecha
pro_fecha_demanda
pro_decreto_admision_tramite_fecha
pro_fecha_copia_sellada
pro_fecha_sentencia_primera_instancia
contract_date
est_costas_arr_cantidad
est_costas_proc
costas_primera_ins
fact_arr
cobrado_arr
importe
pf_importe
intereses


GastosHipotecaYCS
ghs_importe_prestamo
ghs_clausula
ghs_critica_clausula
ghs_aranceles_notariales
ghs_aranceles_registrales
ghs_actos_juridicos
ghs_tasacion
ghs_servicios_gestoria
ghs_tipo_suelo
ghs_cuotas
ghs_fecha_reclamacion
ghs_fecha_comunicacion
ghs_fecha_escritura
ghs_techo
ghs_comunicacion
ghs_banco_prestamo
ghs_deudor_anterior
ghs_nombre_deudor_anterior
ghs_direccion_inmueble
ghs_banco_demanda
ghs_numero_clausula
ghs_titulo_clausula
ghs_extracto_escritura
ghs_condiciones_consumidor
ghs_plazo_amortizacion
ghs_subrogacion
ghs_titulo_escritura
ghs_sucursal
ghs_notario
ghs_ciudad_notario
ghs_banco_demanda_domicilio
ghs_banco_demanda_cif
ghs_protocolo_escritura


ClausulaSuelo
csb_clausula_finalidad
csb_plazo_amortizacion
csb_protocolo_escritura
csb_cuotas_escritura
csb_subrogacion
csb_notario
csb_ciudad_notario
csb_tipo_vivienda
csb_direccion_inmueble
csb_clausulas_cs
csb_oferta_mencion_suelo
csb_texto_cuantia_hipoteca
csb_titulo_clausula_cs
csb_clausula_interes_fijo
csb_titulo_segunda_novacion
csb_tipo_ficha_informacion
csb_tipo_ficha_informacion
csb_nombre_deudor
csb_protocolo_escritura_novacion
csb_texto_sin_comunicacion
csb_texto_breve_segunda_novacion
csb_condiciones_generales
csb_texto_variacion_interes
csb_prestamo_amortizado
csb_texto_breve_primera_novacion
csb_texto_extenso_segunda_novacion
csb_texto_transmision_inmueble
csb_prestamo_credito
csb_demandantes
csb_domicilio_demandantes
csb_dni_demandantes
csb_profesion_demandantes
csb_banco_demanda
csb_banco_prestamo
csb_banco_demanda_cif
csb_banco_demanda_domicilio
csb_titulo_escritura
csb_tipo_deudor
csb_titulo_primera_novacion_cs
csb_clausula_escritura_cs
csb_euribor_minimo
csb_tipo_novacion
csb_tipo_cs
csb_aplica_cs
csb_texto_techo
csb_texto_cs
csb_texto_euribor
csb_texto_periodo_fijo
csb_ciudad_notario_novacion
csb_clausula_escritura_cs_suplico
csb_titulo_clausula_cs_suplico
csb_clausula_oscura
csb_clausula_online
csb_clausula_unicaja
csb_literal_bbva
csb_tipo_novacion_extra
csb_indice
csb_clausula_redondeo
csb_notario_novacion
csb_explicacion_tabla
csb_titulo_escritura_promotor
csb_protocolo_escritura_promotor
csb_clausula_financiera_segunda
csb_texto_periodo_variable
csb_tabla_amortizacion
csb_notario_promotor
csb_texto_comercializacion
csb_texto_extenso_primera_novacion
csb_fecha_comunicacion_previa
csb_fecha_escritura
csb_fecha_vencimiento_credito
csb_fecha_reclamacion
csb_fecha_oferta
csb_fecha_inicio_fijo
csb_fecha_fin_fijo
csb_fecha_segunda_novacion
csb_fecha_primera_novacion
csb_fecha_recibo
csb_fecha_escritura_promotor
csb_fecha_amortizacion_promotor
csb_fecha_tipo_fijo
csb_fecha_primer_acuerdo
csb_fecha_segundo_acuerdo
csb_importe_prestamo
csb_tipo_fijado_suelo
csb_diferencial
csb_nuevo_tipo_fijado
csb_tipo_fijado_techo
csb_importe_reclamacion


BasicCase
eje-auto-despachando-cantidad
cos-ejj-cantidad
costas-ape
costas-eje
pro-decreto-admision-tramite-cantidad
ape-sentencia
cos-ejj-fecha
liq-vistas-fecha
tasa-judicial-fecha
pro-fecha-demanda
pro-fecha-sentencia-primera-instancia
pro-decreto-admision-tramite-fecha
fecha-emision
pro-dc-apelacion
pro-plazo-recurso-hasta
ape-sentencia-firme-desde
pro-contestacion-demanda-fecha
pro-recurso-apelacion
fecha-asignacion
pro-fecha-audiencia
eje-auto-despachando-fecha
cos-po-fecha
cos-apelacion-fecha
pro-fecha-copia-sellada
pro-fecha-juicio
pro-fecha-vistas
pro-audiencia-provincial
cos-apelacion-notas
ape-procedimiento
ciudad-oficina
num-cuenta-banco
incidencias-juicio
procurador-externo
abogado-contrario
tipo-regimen
partido-judicial
procurador-contrario
eje-prodecimiento
pro-ubicacion-poder-notarial
cos-po-notas
ubicacion-hoja-encargo
num-bastanteo
cos-ejj-notas
pro-ciudad-jpj
liq-notas
track
pro-jpl
suceso
nig2
instructions-procesal
persona-caso
eje-notas
pro-autos
docs-fondo-fijos
persona-redactor
persona-completado
persona-comercial
procurador-audiencia
persona-auditor
abogado-vistas
procurador-juicio
abogado-audiencia
ubicacion-procurador
persona-ejecuciones
abogado-juicio
procurador-vistas
juicio-verbal
pro-visto-sentencia
pro-no-vista-aprobada
pro-no-vista
condena-costas-apelacion
condena-costas-primera
mifid
poder
historico
contrato
orden
acepta-la-firma
carta-extrajudicial
hoja-encargo-firmada
liq-vistas-cantidad
tasa-judicial-cantidad
cos-po-cantidad
cos-apelacion-cantidad
pro-porcentaje-hoja-encargo


GastosHipoteca
gh_arancel_extra
gh_tasacion_extra
gh_transmisiones_extra
gh_servicios_gestoria_extra
gh_actos_juridicos_extra
gh_aranceles_notariales_extra
gh_prestamo_amortizado
gh_protocolo_escritura_extra
gh_clausula_oscura
gh_compra_venta
gh_texto_autorizacion
gh_texto_clausula
gh_texto_clausula_escritura_extra
gh_texto_gestor_comparecen
gh_texto_gestoria
gh_tipo_vivienda
gh_titulo_clausula_extra
gh_titulo_clausula_escritura_extra
gh_titulo_escritura_extra
gh_vencimiento_anticipado
gh_vencimiento_tipo
gh_vencimiento_anticipado_extra
gh_vencimiento_tipo_extra
gh_prestamo_credito
gh_plazo_amortizacion
gh_condiciones_generales
gh_impago_incumplimiento
gh_texto_autorizacion
gh_texto_clausula
gh_texto_clausula_escritura_extra
gh_texto_gestor_comparecen
gh_texto_gestoria
gh_importe_prestamo
gh_subrogacion
gh_titulo_escritura
gh_numero_clausula
gh_protocolo_escritura
gh_banco_prestamo
gh_ciudad_notario
gh_notario
gh_titulo_clausula
gh_banco_demanda_domicilio
gh_banco_demanda_cif
gh_banco_demanda
gh_jpi_cliente
gh_apartados_no_nulidad
gh_condiciones_generales
gh_dni_demandantes
gh_domicilio_demandantes
gh_impago_incumplimiento
gh_notario_extra
email_responsible
appointment_office
files_count
pro_fecha_audiencia
pro_fecha_juicio
pro_fecha_vistas
created
modified
deleted
pf_fecha
pro_fecha_demanda
pro_decreto_admision_tramite_fecha
pro_fecha_copia_sellada
pro_fecha_sentencia_primera_instancia
contract_date
importe
pf_importe
intereses

ANEXO

Bundles EAV

Class	Bundle	Attribute	Type	Visibility
User	BasicUser	area	string	public
User	BasicUser	departamento	string	public
User	BasicUser	oficina	string	public
User	BasicUser	planta	string	public
User	BasicUser	imei	string	public
User	BasicUser	puk	string	public
User	BasicUser	ordenador	string	public
User	BasicUser	codigo-impresora	string	public
User	BasicUser	extension-1	string	public
User	BasicUser	extension-2	string	public
User	BasicUser	procurador-in	boolean	public
User	BasicUser	abogado-in	boolean	public
User	BasicUser	iface_addr	string	public
Client	BasicClient	empresa	boolean	public
Client	BasicClient	direccion	string	public
Client	BasicClient	ciudad-cliente	string	public
Client	BasicClient	telefono-1	string	public
Client	BasicClient	telefono-2	string	public
Client	BasicClient	codigo-postal	integer	public
Client	BasicClient	empresa-cif	string	public
Client	BasicClient	empresa-nombre-juridico	string	public
Client	BasicClient	consentimiento	boolean	public
Case	GastosHipotecaYCS	ghs_fecha_reclamacion	date	public
Case	GastosHipotecaYCS	ghs_fecha_comunicacion	date	public
Case	GastosHipotecaYCS	ghs_comunicacion	string	public
Case	GastosHipotecaYCS	ghs_techo	string	public
Case	GastosHipotecaYCS	ghs_nombre_deudor_anterior	string	public
Case	GastosHipotecaYCS	ghs_deudor_anterior	string	public
Case	GastosHipotecaYCS	ghs_tipo_suelo	float	public
Case	GastosHipotecaYCS	ghs_numero_clausula	string	public
Case	GastosHipotecaYCS	ghs_titulo_clausula	string	public
Case	GastosHipotecaYCS	ghs_direccion_inmueble	string	public
Case	GastosHipotecaYCS	ghs_extracto_escritura	text	public
Case	GastosHipotecaYCS	ghs_servicios_gestoria	float	public
Case	GastosHipotecaYCS	ghs_tasacion	float	public
Case	GastosHipotecaYCS	ghs_actos_juridicos	float	public
Case	GastosHipotecaYCS	ghs_aranceles_registrales	float	public
Case	GastosHipotecaYCS	ghs_aranceles_notariales	float	public
Case	GastosHipotecaYCS	ghs_critica_clausula	text	public
Case	GastosHipotecaYCS	ghs_clausula	text	public
Case	GastosHipotecaYCS	ghs_condiciones_consumidor	text	public
Case	GastosHipotecaYCS	ghs_cuotas	integer	public
Case	GastosHipotecaYCS	ghs_plazo_amortizacion	string	public
Case	GastosHipotecaYCS	ghs_importe_prestamo	float	public
Case	GastosHipotecaYCS	ghs_subrogacion	string	public

Case	GastosHipotecaYCS	ghs_titulo_escritura	string	public
Case	GastosHipotecaYCS	ghs_protocolo_escritura	string	public
Case	GastosHipotecaYCS	ghs_ciudad_notario	string	public
Case	GastosHipotecaYCS	ghs_notario	string	public
Case	GastosHipotecaYCS	ghs_fecha_escritura	date	public
Case	GastosHipotecaYCS	ghs_sucursal	string	public
Case	GastosHipotecaYCS	ghs_banco_demanda_domicilio	string	public
Case	GastosHipotecaYCS	ghs_banco_demanda_cif	string	public
Case	GastosHipotecaYCS	ghs_banco_demanda	string	public
Case	GastosHipotecaYCS	ghs_banco_prestamo	string	public
Case	GastosHipoteca	gh_servicios_gestoria	float	public
Case	GastosHipoteca	gh_tasacion	float	public
Case	GastosHipoteca	gh_actos_juridicos	float	public
Case	GastosHipoteca	gh_aranceles_registrales	float	public
Case	GastosHipoteca	gh_aranceles_notariales	float	public
Case	GastosHipoteca	gh_plazo_amortizacion	string	public
Case	GastosHipoteca	gh_importe_prestamo	string	public
Case	GastosHipoteca	gh_subrogacion	string	public
Case	GastosHipoteca	gh_titulo_escritura	string	public
Case	GastosHipoteca	gh_numero_clausula	string	public
Case	GastosHipoteca	gh_protocolo_escritura	string	public
Case	GastosHipoteca	gh_banco_prestamo	string	public
Case	GastosHipoteca	gh_ciudad_notario	string	public
Case	GastosHipoteca	gh_notario	string	public
Case	GastosHipoteca	gh_fecha_escritura	date	public
Case	GastosHipoteca	gh_titulo_clausula	string	public
Case	GastosHipoteca	gh_banco_demanda_domicilio	string	public
Case	GastosHipoteca	gh_banco_demanda_cif	string	public
Case	GastosHipoteca	gh_banco_demanda	string	public
Case	**GastosHipoteca**	**gh_jpi_cliente**	**string**	**public**
Case	GastosHipoteca	gh_actos_juridicos_extra	float	public
Case	GastosHipoteca	gh_apartados_no_nulidad	string	public
Case	GastosHipoteca	gh_aranceles_notariales_extra	float	public
Case	GastosHipoteca	gh_aranceles_registrales_extra	float	public
Case	GastosHipoteca	gh_condiciones_generales	text	public
Case	GastosHipoteca	gh_demandantes	string	public
Case	GastosHipoteca	gh_dni_demandantes	string	public
Case	GastosHipoteca	gh_domicilio_demandantes	string	public
Case	GastosHipoteca	gh_fecha_escritura_extra	date	public
Case	GastosHipoteca	gh_impago_incumplimiento	text	public
Case	GastosHipoteca	gh_notario_extra	string	public
Case	GastosHipoteca	gh_prestamo_amortizado	string	public
Case	GastosHipoteca	gh_protocolo_escritura_extra	string	public
Case	GastosHipoteca	gh_servicios_gestoria_extra	float	public
Case	GastosHipoteca	gh_tasacion_extra	float	public
Case	GastosHipoteca	gh_texto_autorizacion	text	public

Case	GastosHipoteca	gh_texto_clausula	text	public
Case	GastosHipoteca	gh_texto_clausula_escritura_extra	text	public
Case	GastosHipoteca	gh_texto_gestor_comparecen	text	public
Case	GastosHipoteca	gh_texto_gestoria	text	public
Case	GastosHipoteca	gh_tipo_vivienda	string	public
Case	GastosHipoteca	gh_titulo_clausula_extra	string	public
Case	GastosHipoteca	gh_titulo_clausula_escritura_extra	string	public
Case	GastosHipoteca	gh_titulo_escritura_extra	string	public
Case	GastosHipoteca	gh_vencimiento_anticipado	string	public
Case	GastosHipoteca	gh_vencimiento_tipo	string	public
Case	GastosHipoteca	gh_vencimiento_anticipado_extra	string	public
Case	GastosHipoteca	gh_vencimiento_tipo_extra	string	public
Case	GastosHipoteca	gh_prestamo_credito	string	public
Case	GastosHipoteca	gh_fecha_vencimiento_credito	date	public
Case	GastosHipoteca	gh_transmisiones	float	public
Case	GastosHipoteca	gh_transmisiones_extra	float	public
Case	GastosHipoteca	gh_compra_venta	string	public
Case	GastosHipoteca	gh_clausula_oscura	string	public
Case	ClausulaSuelo	csb_prestamo_credito	string	public
Case	ClausulaSuelo	csb_demandantes	string	public
Case	ClausulaSuelo	csb_domicilio_demandantes	string	public
Case	ClausulaSuelo	csb_dni_demandantes	string	public
Case	ClausulaSuelo	csb_profesion_demandantes	string	public
Case	ClausulaSuelo	csb_banco_demanda	string	public
Case	ClausulaSuelo	csb_banco_prestamo	string	public
Case	ClausulaSuelo	csb_banco_demanda_cif	string	public
Case	ClausulaSuelo	csb_banco_demanda_domicilio	string	public
Case	ClausulaSuelo	csb_titulo_escritura	string	public
Case	ClausulaSuelo	csb_fecha_escritura	date	public
Case	ClausulaSuelo	csb_protocolo_escritura	string	public
Case	ClausulaSuelo	csb_importe_prestamo	float	public
Case	ClausulaSuelo	csb_plazo_amortizacion	string	public
Case	ClausulaSuelo	csb_cuotas_escritura	string	public
Case	ClausulaSuelo	csb_subrogacion	string	public
Case	ClausulaSuelo	csb_notario	string	public
Case	ClausulaSuelo	csb_ciudad_notario	string	public
Case	ClausulaSuelo	csb_fecha_vencimiento_credito	date	public
Case	ClausulaSuelo	csb_tipo_vivienda	string	public
Case	ClausulaSuelo	csb_direccion_inmueble	string	public
Case	ClausulaSuelo	csb_tipo_deudor	string	public
Case	ClausulaSuelo	csb_nombre_deudor	string	public
Case	ClausulaSuelo	csb_fecha_reclamacion	date	public
Case	ClausulaSuelo	csb_clausula_finalidad	string	public
Case	ClausulaSuelo	csb_clausulas_cs	string	public
Case	ClausulaSuelo	csb_tipo_cs	string	public
Case	ClausulaSuelo	csb_clausula_escritura_cs	string	public

Case	ClausulaSuelo	csb_titulo_clausula_cs	string	public
Case	ClausulaSuelo	csb_tipo_fijado_suelo	float	public
Case	ClausulaSuelo	csb_aplica_cs	string	public
Case	ClausulaSuelo	csb_fecha_oferta	date	public
Case	ClausulaSuelo	csb_oferta_mencion_suelo	string	public
Case	ClausulaSuelo	csb_clausula_financiera_segunda	text	public
Case	ClausulaSuelo	csb_clausula_interes_fijo	string	public
Case	ClausulaSuelo	csb_fecha_inicio_fijo	date	public
Case	ClausulaSuelo	csb_fecha_fin_fijo	date	public
Case	ClausulaSuelo	csb_texto_periodo_fijo	text	public
Case	ClausulaSuelo	csb_diferencial	float	public
Case	ClausulaSuelo	csb_texto_periodo_variable	text	public
Case	ClausulaSuelo	csb_texto_euribor	text	public
Case	ClausulaSuelo	csb_texto_cs	text	public
Case	ClausulaSuelo	csb_euribor_minimo	string	public
Case	ClausulaSuelo	csb_tabla_amortizacion	text	public
Case	ClausulaSuelo	csb_texto_comercializacion	text	public
Case	ClausulaSuelo	csb_fecha_comunicacion_previa	date	public
Case	ClausulaSuelo	csb_fecha_recibo	date	public
Case	ClausulaSuelo	csb_texto_sin_comunicacion	text	public
Case	ClausulaSuelo	csb_condiciones_generales	text	public
Case	ClausulaSuelo	csb_texto_variacion_interes	text	public
Case	ClausulaSuelo	csb_titulo_primera_novacion	string	public
Case	ClausulaSuelo	csb_fecha_primera_novacion	date	public
Case	ClausulaSuelo	csb_texto_breve_primera_novacion	text	public
Case	ClausulaSuelo	csb_texto_extenso_primera_novacion	text	public
Case	ClausulaSuelo	csb_titulo_segunda_novacion	string	public
Case	ClausulaSuelo	csb_fecha_segunda_novacion	date	public
Case	ClausulaSuelo	csb_texto_breve_segunda_novacion	text	public
Case	ClausulaSuelo	csb_texto_extenso_segunda_novacion	text	public
Case	ClausulaSuelo	csb_prestamo_amortizado	string	public
Case	ClausulaSuelo	csb_fecha_amortizacion	date	public
Case	ClausulaSuelo	csb_tipo_ficha_informacion	string	public
Case	ClausulaSuelo	csb_tipo_novacion	string	public
Case	ClausulaSuelo	csb_protocolo_escritura_novacion	string	public
Case	ClausulaSuelo	csb_notario_novacion	string	public
Case	ClausulaSuelo	csb_ciudad_notario_novacion	string	public
Case	ClausulaSuelo	csb_nuevo_tipo_fijado_suelo	float	public
Case	ClausulaSuelo	csb_fecha_tipo_fijo	date	public
Case	ClausulaSuelo	csb_texto_transmision_inmueble	text	public
Case	ClausulaSuelo	csb_texto_cuantia_hipoteca	text	public
Case	ClausulaSuelo	csb_clausula_escritura_cs_suplico	string	public
Case	ClausulaSuelo	csb_titulo_clausula_cs_suplico	string	public
Case	ClausulaSuelo	csb_clausula_oscura	string	public
Case	ClausulaSuelo	csb_clausula_online	string	public
Case	ClausulaSuelo	csb_clausula_unicaja	string	public

Case	ClausulaSuelo	csb_texto_techo	text	public
Case	ClausulaSuelo	csb_tipo_fijado_techo	float	public
Case	ClausulaSuelo	csb_literal_bbva	string	public
Case	ClausulaSuelo	csb_importe_reclamacion	float	public
Case	ClausulaSuelo	csb_explicacion_tabla	text	public
Case	ClausulaSuelo	csb_tipo_novacion_extra	string	public
Case	ClausulaSuelo	csb_fecha_primer_acuerdo	date	public
Case	ClausulaSuelo	csb_fecha_segundo_acuerdo	date	public
Case	ClausulaSuelo	csb_indice	string	public
Case	ClausulaSuelo	csb_clausula_redondeo	string	public
Case	ClausulaSuelo	csb_titulo_escritura_promotor	string	public
Case	ClausulaSuelo	csb_fecha_escritura_promotor	date	public
Case	ClausulaSuelo	csb_notario_promotor	string	public
Case	ClausulaSuelo	csb_protocolo_escritura_promotor	string	public
Case	BasicCase	ciudad-oficina	string	public
Case	BasicCase	fecha-asignacion	date	public
Case	BasicCase	persona-caso	string	public
Case	BasicCase	fecha-emision	date	public
Case	BasicCase	mifid	boolean	public
Case	BasicCase	acepta-la-firma	boolean	public
Case	BasicCase	orden	boolean	public
Case	BasicCase	contrato	boolean	public
Case	BasicCase	historico	boolean	public
Case	BasicCase	poder	boolean	public
Case	BasicCase	carta-extrajudicial	boolean	public
Case	BasicCase	tasa-judicial-fecha	date	public
Case	BasicCase	tasa-judicial-cantidad	float	public
Case	BasicCase	pro-porcentaje-hoja-encargo	float	public
Case	BasicCase	pro-fecha-demanda	date	public
Case	BasicCase	pro-decreto-admision-tramite-fecha	date	public
Case	BasicCase	pro-decreto-admision-tramite-cantidad	float	public
Case	BasicCase	pro-autos	string	public
Case	BasicCase	pro-jpi	string	public
Case	BasicCase	pro-fecha-audiencia	datetime	public
Case	BasicCase	pro-fecha-juicio	datetime	public
Case	BasicCase	pro-fecha-sentencia-primera-instancia	date	public
Case	BasicCase	pro-plazo-recurso-hasta	date	public
Case	BasicCase	pro-recurso-apelacion	date	public
Case	BasicCase	pro-do-apelacion	date	public
Case	BasicCase	pro-audiencia-provincial	string	public
Case	BasicCase	ape-procedimiento	string	public
Case	BasicCase	ape-sentencia	date	public
Case	BasicCase	ape-sentencia-firme-desde	date	public
Case	BasicCase	pro-fecha-copia-sellada	date	public
Case	BasicCase	eje-auto-despachando-fecha	date	public
Case	BasicCase	eje-auto-despachando-cantidad	float	public

Case	BasicCase	eje-notas	string	public
Case	BasicCase	eje-prodecimiento	string	public
Case	BasicCase	liq-vistas-fecha	date	public
Case	BasicCase	liq-vistas-cantidad	float	public
Case	BasicCase	liq-notas	string	public
Case	BasicCase	cos-po-fecha	date	public
Case	BasicCase	cos-po-cantidad	float	public
Case	BasicCase	cos-po-notas	string	public
Case	BasicCase	cos-apelacion-fecha	date	public
Case	BasicCase	cos-apelacion-cantidad	float	public
Case	BasicCase	cos-apelacion-notas	string	public
Case	BasicCase	cos-etj-fecha	date	public
Case	BasicCase	cos-etj-cantidad	float	public
Case	BasicCase	cos-etj-notas	string	public
Case	BasicCase	pro-ciudad-jpi	string	public
Case	BasicCase	hoja-encargo-firmada	boolean	public
Case	BasicCase	pro-ubicacion-poder-notarial	string	public
Case	BasicCase	partido-judicial	string	public
Case	BasicCase	suceso	string	public
Case	BasicCase	tipo-regimen	string	public
Case	BasicCase	ubicacion-hoja-encargo	string	public
Case	BasicCase	ubicacion-procurador	integer	public
Case	BasicCase	track	string	public
Case	BasicCase	abogado-audiencia	integer	public
Case	BasicCase	abogado-juicio	integer	public
Case	BasicCase	procurador-audiencia	integer	public
Case	BasicCase	procurador-juicio	integer	public
Case	BasicCase	abogado-vistas	integer	public
Case	BasicCase	procurador-vistas	integer	public
Case	BasicCase	pro-fecha-vistas	datetime	public
Case	BasicCase	pro-contestacion-demanda-fecha	date	public
Case	BasicCase	num-bastanteo	string	public
Case	BasicCase	num-cuenta-banco	string	public
Case	BasicCase	incidencias-juicio	string	public
Case	BasicCase	juicio-verbal	boolean	public
Case	BasicCase	persona-comercial	integer	public
Case	BasicCase	persona-completado	integer	public
Case	BasicCase	persona-auditor	integer	public
Case	BasicCase	persona-redactor	integer	public
Case	BasicCase	procurador-contrario	string	public
Case	BasicCase	condena-costas-primera	boolean	public
Case	BasicCase	condena-costas-apelacion	boolean	public
Case	BasicCase	pro-no-vista	boolean	public
Case	BasicCase	pro-no-vista-aprobada	boolean	public
Case	BasicCase	pro-visto-sentencia	boolean	public
Case	BasicCase	procurador-externo	string	public

Case	BasicCase	costas-ape	float	public
Case	BasicCase	costas-eje	float	public
Case	BasicCase	persona-ejecuciones	integer	public
Case	BasicCase	abogado-contrario	string	public
Case	BasicCase	docs-fondo-files	text	public
Case	BasicCase	nig2	string	public
Case	BasicCase	instructions-procesal	text	public

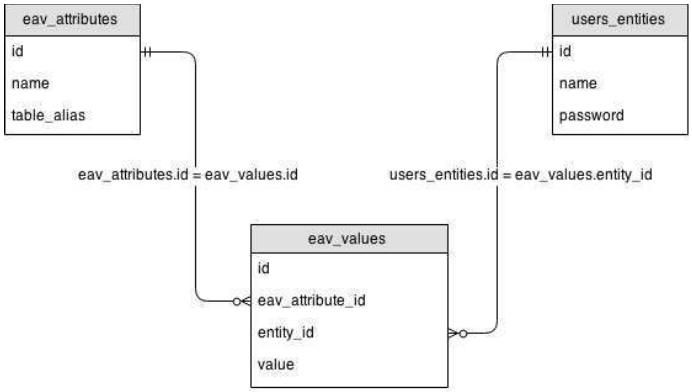
ANEXO

EAV Plugin

```
<ul>
    <?php foreach ($articles as $article): ?>
    <li><?php $article->get('title'); ?></li>
    <?php endforeach; ?>
</ul>
```

EAV API

> Entity–attribute–value model (EAV) is a data model to describe entities where the number of attributes (properties, parameters) that can be used to describe them is potentially vast, but the number that will actually apply to a given entity is relatively modest. In mathematics, this model is known as a sparse matrix. EAV is also known as object–attribute–value model, vertical database model and open schema.
>
> —Wikipedia



You will typically use an EAV pattern when you have Entities with a variable number of attributes, and these attributes can be of different types. This makes it impossible to define these attributes as column in the entity's table, because there would be too many, most of them will not have data, and you can't deal with dynamic attributes at all because columns need to be pre-defined in relational databases.

To solve this situation in a relational fashion you would create a child table, and relate that to the 'entity' table using a One-to-Many relation, where every attribute would become a record in the child table ("eav_attributes" in the image). Downside of this approach however is that to be able to get a specific attribute value, you'll have to loop over all related records, compare the value of the attribute column with the attribute you look for, and if a match is found, get the contents of the value column.

QuickAppsCMS's EAV API uses this same implementation, but allows you to merge these virtual attributes with the entity, so the attributes become properties of the entity object.

Table Of Contents

Usage

To use the EAV API you must attach the `Eav.Eav` behavior to the table you wish to "extend", for example:

```
use Cake\ORM\Table;

class UsersTable extends Table
{
    public function initialize(Table $table)
    {
        $this->addBehavior('Eav.Eav');
    }
}
```

Defining Attributes

Once EAV behavior is attached to your table, you can now start defining virtual columns. There are two ways of defining virtual columns, CLI based or php-script based. We'll explain how to define such columns using both methods.

Using EAV CLI (Recommended)

EAV plugin provides a simple management command-line-interface (CLI) which allows you to easily add or drop virtual columns.

You need to tell which table is being altered, what action you wish to perform (add new virtual column, or drop existing one). And if you are adding new column you must provide column information (column name, data type, etc). Below an example on how to add new virtual column named *user_age*:

```
user@name:/path/to/bin/$ cake Eav.table schema --use UsersPlugin.UsersTable --
↪action add --name user_age --type integer --searchable
```

The `searchable` indicates that this virtual column can be in `WHERE` clauses. If you want to drop an existing column:

```
user@name:/path/to/bin/$ cake Eav.table schema --use UsersPlugin.UsersTable --
↪action drop --name user_age
```

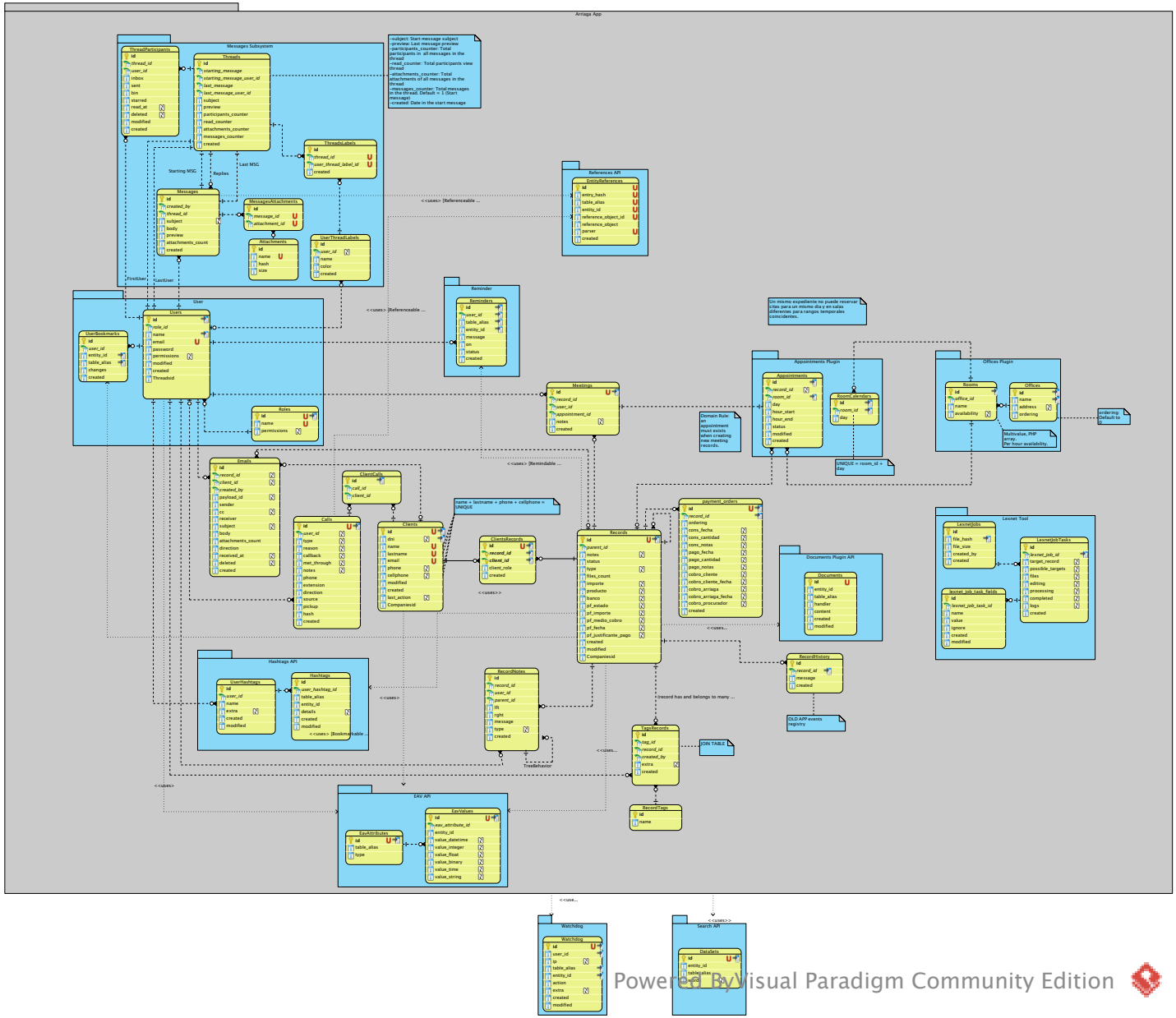
Check EAV CLI help for more options available.

Using PHP Script

> **Warning:** You should do this step just once, otherwise you will end unnecessary updating columns every time the script is executed.

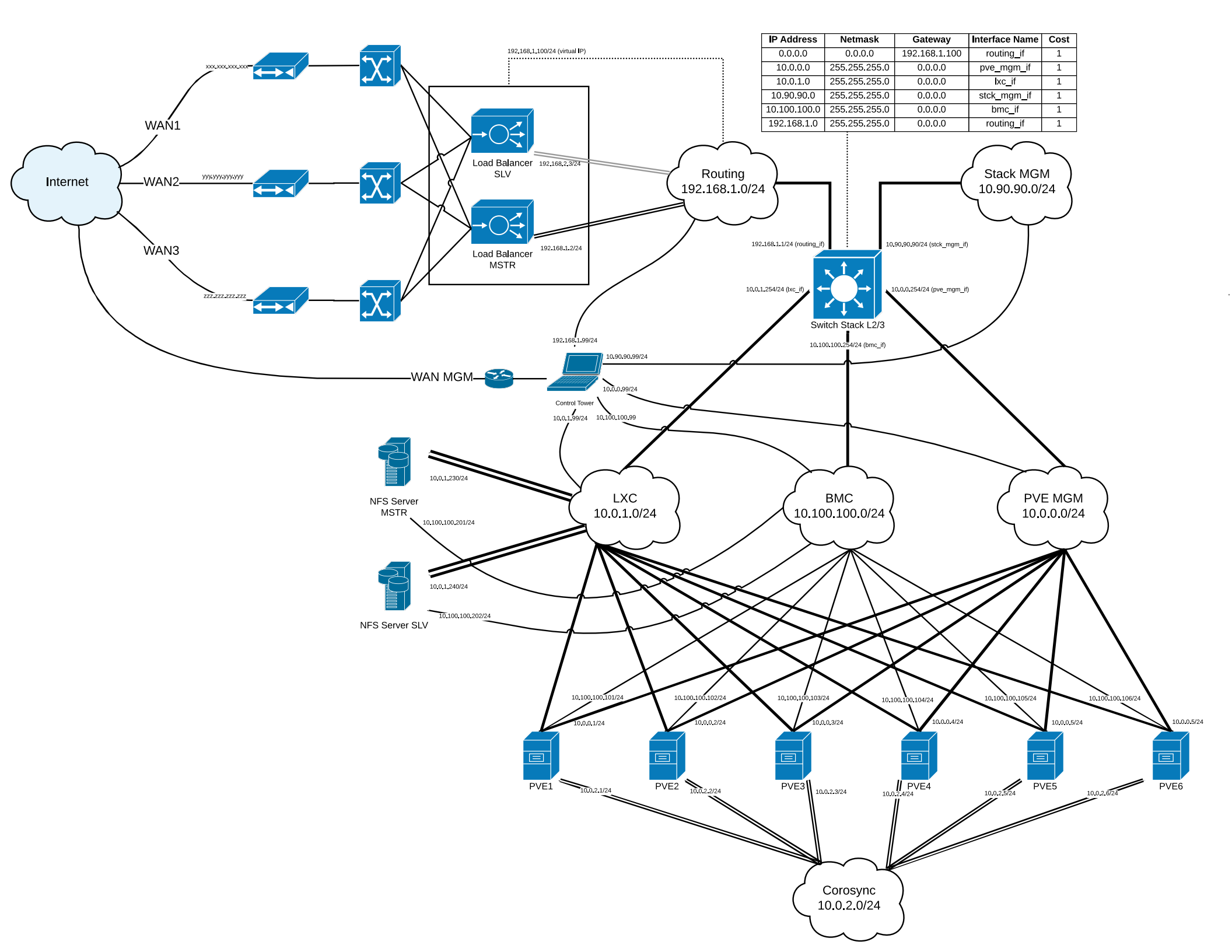
ANEXO

Modelo Relacional



ANEXO

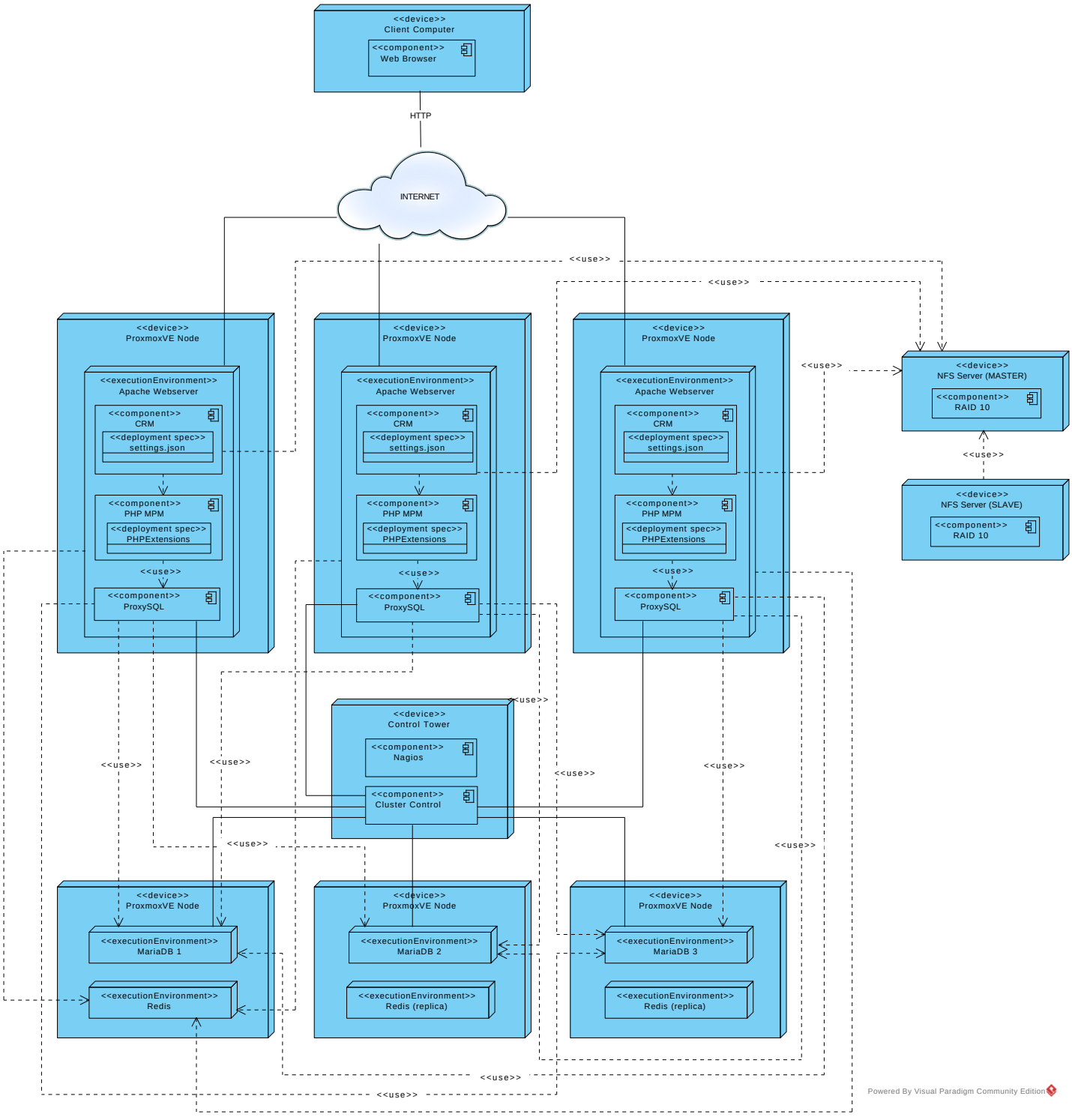
Diagrama de Red



IP Address	Netmask	Gateway	Interface Name	Cost
0.0.0.0	0.0.0.0	192.168.1.100	routing_if	1
10.0.0.0	255.255.255.0	0.0.0.0	pve_mgm_if	1
10.0.1.0	255.255.255.0	0.0.0.0	lxc_if	1
10.90.90.0	255.255.255.0	0.0.0.0	stck_mgm_if	1
10.100.100.0	255.255.255.0	0.0.0.0	bmc_if	1
192.168.1.0	255.255.255.0	0.0.0.0	routing_if	1

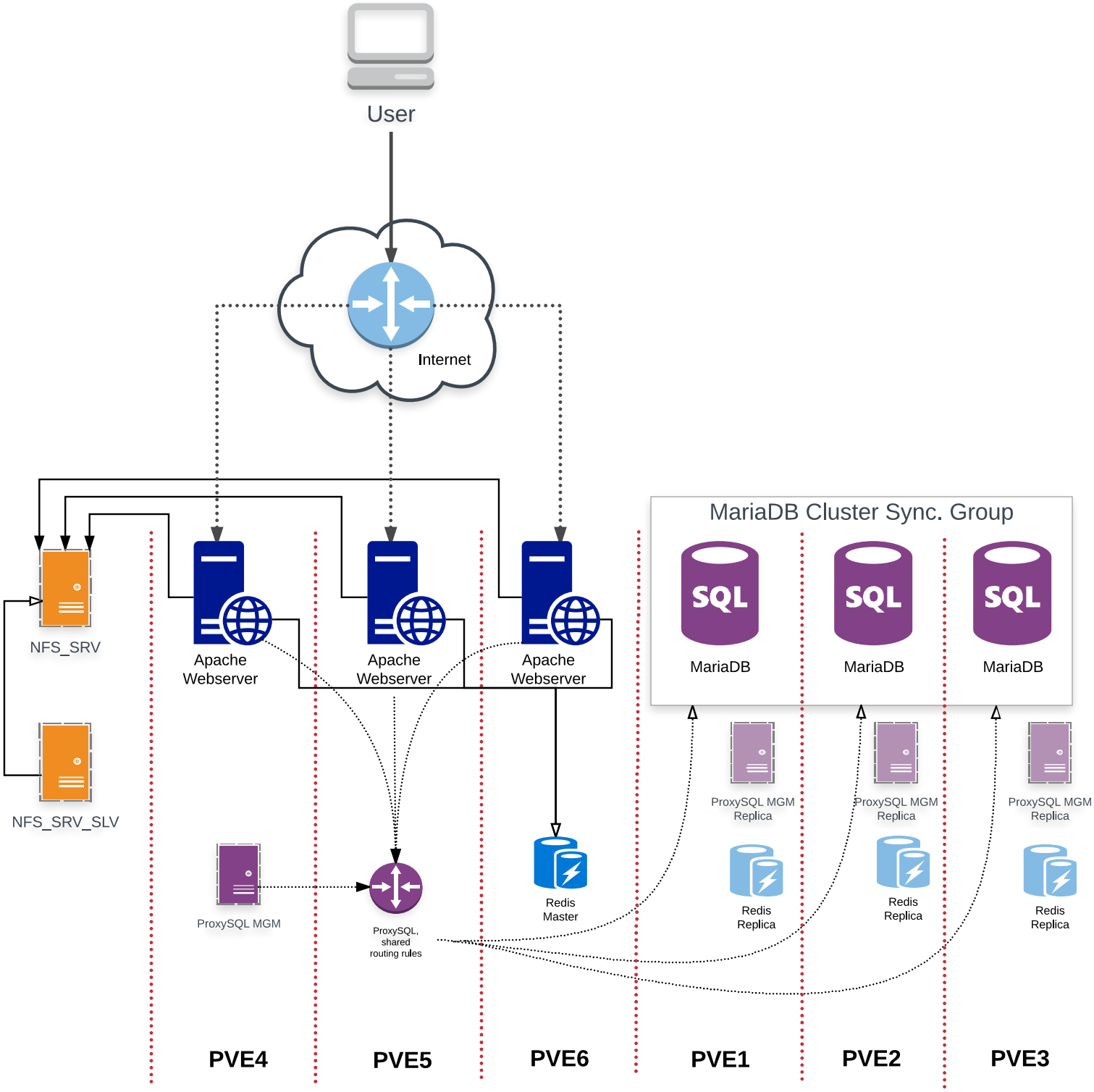
ANEXO

Diagrama de Despliegue


<<device>>
Client Computer
<<component>>
Web Browser
HTTP
INTERNET
<<use>>
<<device>>
ProxmoxVE Node
<<executionEnvironment>>
Apache Webserver
<<component>>
CRM
<<deployment spec>>
settings.json
<<component>>
PHP MPM
<<deployment spec>>
PHPExtensions
<<use>>
<<component>>
ProxySQL
<<device>>
NFS Server (MASTER)
<<component>>
RAID 10
<<device>>
NFS Server (SLAVE)
<<component>>
RAID 10
<<device>>
Control Tower
<<component>>
Nagios
<<component>>
Cluster Control
<<executionEnvironment>>
MariaDB 1
<<executionEnvironment>>
Redis
<<executionEnvironment>>
MariaDB 2
<<executionEnvironment>>
Redis (replica)
<<executionEnvironment>>
MariaDB 3
<<executionEnvironment>>
Redis (replica)
Powered By Visual Paradigm Community Edition

ANEXO

Relación Componentes Físicos y Virtuales


User
Internet
NFS_SRV
NFS_SRV_SLV
Apache Webserver
Apache Webserver
Apache Webserver
MariaDB Cluster Sync. Group
SQL
MariaDB
SQL
MariaDB
SQL
MariaDB
ProxySQL MGM Replica
ProxySQL MGM Replica
ProxySQL MGM Replica
Redis Replica
Redis Replica
Redis Replica
ProxySQL MGM
ProxySQL, shared routing rules
Redis Master
PVE4
PVE5
PVE6
PVE1
PVE2
PVE3

ANEXO

Relación Hardware y Software

Nombre	Tipo	Modelo	Conf. Red	OS	Puertos Expuestos	Descripción	Ubicación
PVE1	Físico	Asus RS700-E8-RS4 V2 : 32 x Intel Xeon E5-2620 v4 @ 2.10GHz, 128G RAM ECC, 3TB SSD RAID10	ip: 10.0.0.1/24,gw:10.0.0.254	ProxmoxVE 5.0b2	tcp: 22 (ssh), 8006 (proxmox web-ui)	Nodo de virtualización, servicios web	Planta Baja
PVE2	Físico	Asus RS700-E8-RS4 V2 : 32 x Intel Xeon E5-2620 v4 @ 2.10GHz, 128G RAM ECC, 3TB SSD RAID10	ip: 10.0.0.2/24,gw:10.0.0.254	ProxmoxVE 5.0b2	tcp: 22 (ssh), 8006 (proxmox web-ui)	Nodo de virtualización, servicios web	Planta Baja
PVE3	Físico	Asus RS700-E8-RS4 V2 : 32 x Intel Xeon E5-2620 v4 @ 2.10GHz, 128G RAM ECC, 3TB SSD RAID10	ip: 10.0.0.3/24,gw:10.0.0.254	ProxmoxVE 5.0b2	tcp: 22 (ssh), 8006 (proxmox web-ui)	Nodo de virtualización, servicios web	Planta Baja
PVE4	Físico	Asus RS700-E8-RS4 V2 : 80 x Intel Xeon E5-2698 v4 @ 2.20GHz, 512G RAM, 3TB SSD RAID10	ip: 10.0.0.4/24,gw:10.0.0.254	ProxmoxVE 5.0b2	tcp: 22 (ssh), 8006 (proxmox web-ui)	Nodo de virtualización, bases de datos	Planta Baja
PVE5	Físico	Asus RS700-E8-RS4 V2 : 80 x Intel Xeon E5-2698 v4 @ 2.20GHz, 512G RAM, 3TB SSD RAID10	ip: 10.0.0.5/24,gw:10.0.0.254	ProxmoxVE 5.0b2	tcp: 22 (ssh), 8006 (proxmox web-ui)	Nodo de virtualización, bases de datos	Planta Baja
PVE6	Físico	Asus RS700-E8-RS4 V2 : 80 x Intel Xeon E5-2698 v4 @ 2.20GHz, 512G RAM, 3TB SSD RAID10	ip: 10.0.0.6/24,gw:10.0.0.254	ProxmoxVE 5.0b2	tcp: 22 (ssh), 8006 (proxmox web-ui)	Nodo de virtualización, bases de datos y laboratorio	Planta Baja
NFS	Físico	HP Proliant DL380p Gen8 : 12 x Intel Xeon E5-2620 v2 @ 2.10GHz, 32G RAM ECC, 18TB HDD RAID 10	ip: 10.0.0.10/24,gw:10.0.0.254	ProxmoxVE 5.0b5	tcp: 22 (ssh)	Servicios NFS (delimitados a 10.0.0.0/23)	Planta Baja
MGM	Físico	Custom	WAN IP fija 4G	Windows10 Pro	tcp: 3389 (remote desk)	Gestión y monitorización de infraestructura	Planta Baja
Apache1	LXC	(virtual)	ip: 10.0.1.101/24,gw:10.0.1.254	Ubuntu 16.04	tcp: 80 (http), 443 (https)	Servicios web CRM	PVE1
Apache2	LXC	(virtual)	ip: 10.0.1.102/24,gw:10.0.1.254	Ubuntu 16.04	tcp: 80 (http), 443 (https)	Servicios web CRM	PVE2
Apache3	LXC	(virtual)	ip: 10.0.1.103/24,gw:10.0.1.254	Ubuntu 16.04	tcp: 80 (http), 443 (https)	Servicios web CRM	PVE3
Maria1	LXC	(virtual)	ip: 10.0.1.201/24,gw:10.0.1.254	Ubuntu 16.04	tcp: 22 (ssh), 3306 (mysql), 4568 (galera), 4444 (galera), tcp/udp: 4567 (galera)	Nodo MariaDB Cluster	PVE4
Maria2	LXC	(virtual)	ip: 10.0.1.202/24,gw:10.0.1.254	Ubuntu 16.04	tcp: 22 (ssh), 3306 (mysql), 4568 (galera), 4444 (galera), tcp/udp: 4567 (galera)	Nodo MariaDB Cluster	PVE5
Maria3	LXC	(virtual)	ip: 10.0.1.203/24,gw:10.0.1.254	Ubuntu 16.04	tcp: 22 (ssh), 3306 (mysql), 4568 (galera), 4444 (galera), tcp/udp: 4567 (galera)	Nodo MariaDB Cluster	PVE6
Cluster Control	LXC	(virtual)	ip: 10.0.1.200/24,gw:10.0.1.254	Debian 8	tcp: 22 (ssh), 80 (http), 443 (https), 9500 (cmon)	MariaDB Management GUI (ClusterControl by severalnines)	PVE4
ProxySQL	LXC	(virtual)	ip: 10.0.1.210/24,gw:10.0.1.254	Debian 8	tcp: 22 (ssh), 3306 (mysql)	MariaDB Load Balancer	PVE5
Jenkins	LXC	(virtual)	ip: 10.0.1.199/24,gw:10.0.1.254	Ubuntu 16.04	tcp: 22 (ssh), 443 (https)	CI	PVE6
Sandbox	LXC	(virtual)	ip: 10.0.1.190/24,gw:10.0.1.254	Ubuntu 16.04	tcp: 22 (ssh), 443 (https), 80 (http)	Sandbox CRM	PVE6

ANEXO

Template Historias de Usuario

REQUISITO

SOLICITUD

`Indique quién solicita el requisito, incluyendo la fecha en la que se solicitó, la forma de contacto y el mensaje original.`

CONTEXTO

`Describa el contexto del issue, la situación actual en la que este transcurre. Intente proporcionar información suficiente como para poder ubicarse y entender fácilmente el ámbito del requisito solicitado.`

HISTORIA DE USUARIO

`Describa la historia de usuario. Recuerde indicar QUIÉN quiere algo, QUÉ quiere y POR QUÉ lo quiere.`

ARTEFACTOS

`Para dejar constancia, adjunte la solicitud inicial del usuario. De haberlos, adjunte también los demás artefactos asociados el issue.`

INFORMACIÓN ADICIONAL

- Issues relacionados.
- Enlaces de interés.
- etc.

ANEXO

Template Recogida de Incidencias

BUG

REPORTA

`Indique quién reporta la incidencia, incluyendo la fecha en la que se reportó, la forma de contacto y el mensaje original.`

POSIBLES CAUSAS

`Indique en una estimación inicial cuáles son las posibles causas del error.`

REPRODUCCIÓN DEL ERROR

`Describa de la forma más detallada posible los pasos a dar para reproducir el error.`

COMPORTAMIENTO ESPERADO

`Describa cuál era el comportamiento esperado al realizar los pasos previos.`

COMPORTAMIENTO OBSERVADO

`Describa cuál fue el comportamiento que se observó en vez del esperado.`

ARTEFACTOS

`Para dejar constancia, adjunte el informe del usuario que reporta el error. De haberlos, adjunte también los demás artefactos asociados el issue.`

INFORMACIÓN ADICIONAL

- Issues relacionados.
- Enlaces de interés.
- etc.

ANEXO

Backlog

Backlog Agosto 2017

(últimas 100 entradas)

#	Title	Status	Created At	Author	Labels
4781	Refactorizar tratamiento de tags en Records.	OPEN	2017-08-23T09:47:41Z	DanielGSM	prioridad baja, refactorizar, REQ-F, RFC
4779	Añadir etiqueta al crear expediente desde importación web	CLOSED	2017-08-23T08:36:41Z	joseluisitoiz	refactorizar, REQ-F, user story
4778	Fix del render del RecordsHelper	OPEN	2017-08-23T07:48:30Z	AitorGE	en progreso, GUI, prioridad baja, refactorizar, REQ-F, RFC, user story
4777	Cambio en la visualización del listado de expedientes	CLOSED	2017-08-23T06:57:11Z	inakitor	prioridad baja
4772	Fix interfaz resultset row	CLOSED	2017-08-22T10:24:52Z	AitorGE	prioridad baja, REQ-F, user story
4770	error en tabla tags_records del fichero mysql.full.sql	CLOSED	2017-08-22T08:27:24Z	inakitor	bug, prioridad baja, SQL-SCHEMA
4769	Nuevo operador de busqueda para Labels	OPEN	2017-08-22T07:31:53Z	Herberos	prioridad media, REQ-F, user story
4768	CRUD de etiquetas	OPEN	2017-08-22T06:50:59Z	Herberos	GUI, prioridad baja, REQ-F, RFC, user story
4764	Fix interfaz de busqueda de Records	CLOSED	2017-08-21T12:08:19Z	AitorGE	prioridad baja, REQ-F, user story
4758	Notes interfaz	CLOSED	2017-08-21T08:56:39Z	AitorGE	prioridad baja, REQ-F, user story
4756	Bug en el filtrado de etiquetas	CLOSED	2017-08-21T08:44:44Z	joseluisitoiz	bug, GUI, prioridad baja
4755	Herramienta labels masivas	OPEN	2017-08-21T08:43:02Z	Herberos	prioridad baja, REQ-F, user story
4753	Inicializar contadores tabla labels	CLOSED	2017-08-21T07:11:50Z	joseluisitoiz	prioridad baja, REQ-F, SQL-SCHEMA
4752	Añadir etiqueta Oion cuando se contrata o desdobla	CLOSED	2017-08-21T06:56:35Z	inakitor	prioridad baja, REQ-F, user story
4749	Fix resulset row	CLOSED	2017-08-18T10:33:10Z	AitorGE	prioridad baja
4747	Error en el listado de bancos	CLOSED	2017-08-18T10:02:31Z	inakitor	
4739	fix file send & change index records list	CLOSED	2017-08-17T12:42:39Z	inakitor	bug, prioridad baja, refactorizar, REQ-NF, RFC
4737	Bug en las busquedas con el operador de busqueda de etiquetas	CLOSED	2017-08-16T12:57:58Z	Herberos	bug, prioridad baja
4735	Bug seleccionando categorías de etiquetas de expedientes	CLOSED	2017-08-16T12:34:03Z	DanielGSM	bug, GUI, prioridad baja, REQ-NF, resuelto
4734	Refactorización del operador de búsqueda de etiquetas	CLOSED	2017-08-16T12:33:52Z	Herberos	prioridad baja, refactorizar, REQ-F
4730	Eliminación de los permisos en submenú de citas	CLOSED	2017-08-16T11:07:03Z	inakitor	prioridad media, REQ-F, resuelto
4729	Refactorizar etiquetado automágico	OPEN	2017-08-16T10:49:39Z	DanielGSM	prioridad baja, refactorizar, REQ-NF, user story
4728	Cambio orden de grabación de la etiqueta al crear una carta extrajudicial	CLOSED	2017-08-16T10:12:58Z	inakitor	prioridad baja, REQ-F
4722	Fix Citas Directas	CLOSED	2017-08-16T08:18:36Z	AitorGE	prioridad baja
4720	Refactorización operador de busqueda 'Hashtags'	CLOSED	2017-08-14T11:59:31Z	Herberos	prioridad baja, refactorizar, REQ-F
4718	Falta carpeta en expediente	OPEN	2017-08-11T11:22:45Z	joseluisitoiz	maintenance, pregunta, prioridad baja, RFC
4717	Refactorizar accesos al setting "dev-path"	CLOSED	2017-08-11T11:18:32Z	DanielGSM	prioridad baja, refactorizar
4716	Nuevas etiquetas	CLOSED	2017-08-11T11:16:44Z	joseluisitoiz	prioridad baja, REQ-NF, SQL-SCHEMA, user story
4715	Refactorización oficinas en Users	OPEN	2017-08-10T12:57:58Z	Herberos	prioridad media, refactorizar, REQ-NF, RFC, SQL-SCHEMA, user story
4712	Cambio de appointment_office al guardar contract_date	CLOSED	2017-08-10T07:13:00Z	Herberos	prioridad baja, refactorizar, REQ-NF, resuelto, user story
4708	Mostrar información de usuarios dados de baja	CLOSED	2017-08-09T12:32:53Z	Herberos	bug, prioridad media, refactorizar, resuelto
4705	Nueva herramienta de hashtags masivos	CLOSED	2017-08-08T08:40:17Z	Herberos	prioridad media, REQ-F, user story
4693	Error al subir copia sellada en las alertas	CLOSED	2017-08-04T11:15:33Z	Herberos	bug, prioridad alta
4692	Etiqueta, cambio de nombre	CLOSED	2017-08-04T11:14:17Z	jjMonreal	prioridad baja, REQ-NF, SQL-SCHEMA, user story
4687	BBDD 3 nuevos registros en rooms	CLOSED	2017-08-04T08:04:07Z	Yisus-Arri	prioridad baja, REQ-NF, SQL-SCHEMA
4685	Asignación de alertas	CLOSED	2017-08-04T07:34:01Z	ChristopherCastro	bug, prioridad alta, resuelto
4683	Etiqueta nueva Oion	CLOSED	2017-08-03T16:10:26Z	jjMonreal	prioridad baja, REQ-NF, SQL-SCHEMA, user story
4682	Nueva Etiqueta	CLOSED	2017-08-03T13:00:54Z	jjMonreal	prioridad baja, REQ-NF, SQL-SCHEMA, user story
4676	Anular desdoblamiento exp. 170418463	CLOSED	2017-08-03T11:42:49Z	ChristopherCastro	pregunta, prioridad baja, REQ-NF, SQL-SCHEMA
4674	Añadir campo comment a tabla reminders	CLOSED	2017-08-03T11:29:15Z	Herberos	prioridad alta, SQL-SCHEMA
4673	Permisos Rol-Usuario	OPEN	2017-08-03T11:13:24Z	jjMonreal	bug, en progreso, prioridad media
4671	Actualización de permisos mágicos en la Wiki	CLOSED	2017-08-03T08:56:08Z	i-roman	maintenance, pregunta, REQ-NF, resuelto
4670	Permisos a herramienta Importar Docs	CLOSED	2017-08-03T08:54:58Z	jjMonreal	prioridad media, REQ-NF, resuelto, user story
4664	Error en tabla cities con partidos judiciales	OPEN	2017-08-02T07:46:50Z	i-roman	bug, prioridad baja, SQL-SCHEMA
4659	BBDD Añadir 3 registros en rooms	CLOSED	2017-08-01T10:06:53Z	Yisus-Arri	prioridad baja, REQ-NF, SQL-SCHEMA
4657	Cambio en bb.dd. para cambio de json de abogados/procuradores externos	CLOSED	2017-08-01T06:50:23Z	inakitor	Refactor, prioridad baja, REQ-F, SQL-SCHEMA
4648	CRU[D] Oficinas y Salas	OPEN	2017-07-31T07:01:19Z	ChristopherCastro	GUI, prioridad media, REQ-F, RFC, user story
4647	Nuevos usuarios y avatares	OPEN	2017-07-31T06:48:04Z	ChristopherCastro	bug, GUI, prioridad baja
4646	Refactorización front-end	OPEN	2017-07-30T23:03:43Z	ChristopherCastro	en progreso, GUI, prioridad baja, refactorizar, REQ-NF, RFC, user story
4645	Refactorización FS	OPEN	2017-07-30T18:59:35Z	ChristopherCastro	prioridad alta, refactorizar, REQ-NF, RFC, user story
4644	Refactor etiquetas	CLOSED	2017-07-30T18:29:44Z	ChristopherCastro	GUI, prioridad baja, refactorizar, REQ-F, resuelto, user story, ☠ abandonado

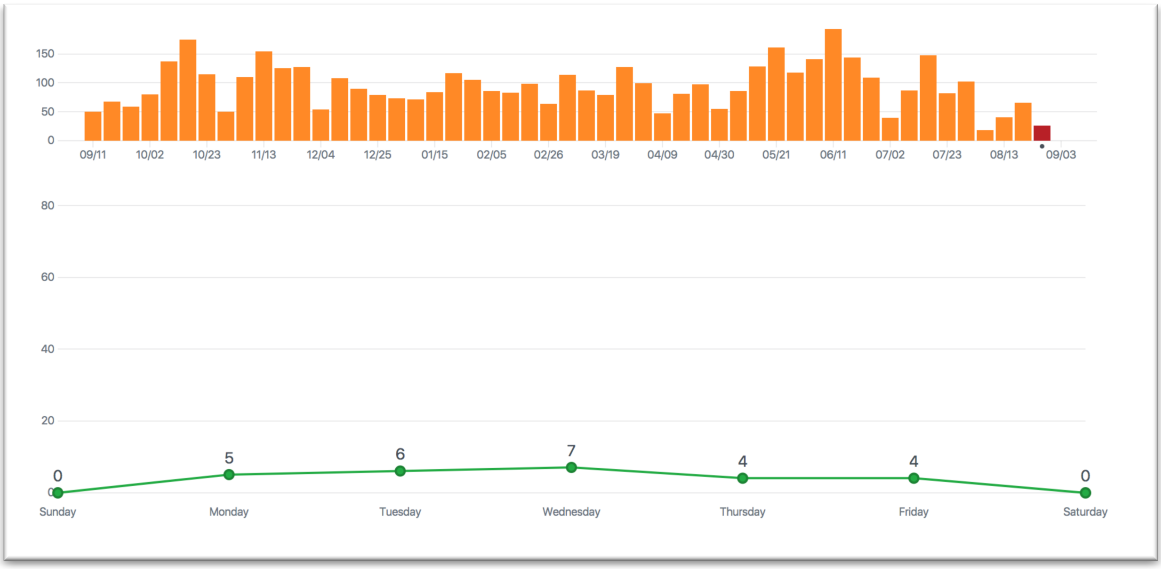
4642	Símbolo `\`	CLOSED	2017-07-28T12:05:36Z	DanielGSM	bug, resuelto, RFC
4640	Nuevo campo virtual	CLOSED	2017-07-28T08:48:40Z	i-roman	inválido, SQL-SCHEMA
4637	Expediente 170104479 - Pestaña de correos no carga	CLOSED	2017-07-28T06:59:31Z	ChristopherCastro	bug, resuelto, RFC
4636	Teléfonos de clientes 91737*	CLOSED	2017-07-28T06:56:23Z	ChristopherCastro	prioridad baja, REQ-NF, resuelto, RFC, SQL-SCHEMA, user story
4625	Actualizar (oootra vez) dirección oficina Almería [urgente]	CLOSED	2017-07-26T10:45:29Z	i-roman	prioridad media, REQ-NF, SQL-SCHEMA
4622	User RulesSet - reglas adicionales y "apagar alertas"	CLOSED	2017-07-26T06:57:21Z	ChristopherCastro	bug, prioridad media, resuelto
4620	Nueva sala para Alicante	CLOSED	2017-07-26T06:30:42Z	i-roman	prioridad media, REQ-NF, SQL-SCHEMA
4612	Petición de campos virtuales	CLOSED	2017-07-25T10:23:36Z	i-roman	prioridad media, REQ-NF, SQL-SCHEMA
4611	Petición de campos virtuales	CLOSED	2017-07-25T10:10:16Z	i-roman	
4608	Actualizar dirección de oficina Almería	CLOSED	2017-07-25T09:10:48Z	i-roman	prioridad media, REQ-NF, SQL-SCHEMA
4604	Cambios en EAV de bb.dd. responsable de redacción	CLOSED	2017-07-24T12:26:50Z	inakitor	inválido, REQ-NF
4602	Necesaria nueva etiqueta "Extrajudicial Firmada"	CLOSED	2017-07-24T11:35:42Z	DanielGSM	prioridad media, REQ-NF, SQL-SCHEMA
4594	BBDD Dos nuevos registros en Rooms	CLOSED	2017-07-21T09:57:16Z	Yisus-Arri	prioridad media, SQL-SCHEMA
4591	Renombrar Cliente Banco	CLOSED	2017-07-21T07:42:59Z	i-roman	prioridad media, SQL-SCHEMA
4579	Dígitos en nombres de usuario	CLOSED	2017-07-20T08:41:35Z	ChristopherCastro	prioridad media, REQ-NF, resuelto, user story
4577	BBDD Cambio en Cities - 4 Registros	CLOSED	2017-07-19T13:07:17Z	Yisus-Arri	SQL-SCHEMA
4561	Nueva etiqueta	CLOSED	2017-07-18T12:26:58Z	ChristopherCastro	resuelto, SQL-SCHEMA
4557	BBDD Nueva Sala en Alicante	CLOSED	2017-07-18T11:06:48Z	Yisus-Arri	SQL-SCHEMA
4556	BBDD Nueva oficina en Guadalajara	CLOSED	2017-07-18T10:26:21Z	Yisus-Arri	REQ-NF, SQL-SCHEMA
4547	Campo virtual nuevo	CLOSED	2017-07-17T12:32:59Z	i-roman	SQL-SCHEMA
4524	Desdoble de expedientes duplica información	CLOSED	2017-07-13T08:59:54Z	ChristopherCastro	bug, prioridad media, resuelto
4522	Fix atributo virtual	CLOSED	2017-07-13T06:55:24Z	i-roman	bug, resuelto, SQL-SCHEMA
4519	Creación de una oficina virtual para atender citas online	CLOSED	2017-07-12T13:34:54Z	DanielGSM	prioridad media, REQ-NF, resuelto, SQL-SCHEMA
4511	Nuevos atributos virtuales	CLOSED	2017-07-12T10:12:16Z	ChristopherCastro	REQ-NF, resuelto, SQL-SCHEMA
4499	Nuevas oficinas y salas	CLOSED	2017-07-11T12:36:28Z	ChristopherCastro	REQ-NF, resuelto, SQL-SCHEMA
4498	Nuevo atributo EAV	CLOSED	2017-07-11T12:28:45Z	ChristopherCastro	REQ-NF, resuelto, SQL-SCHEMA
4485	Creación de nuevas etiquetas	CLOSED	2017-07-10T09:46:01Z	joseluisitoiz	REQ-NF, resuelto, SQL-SCHEMA
4472	Objetos perdidos	CLOSED	2017-07-06T21:31:16Z	DanielGSM	maintenance, prioridad alta, resuelto
4453	Migración a GC	CLOSED	2017-06-30T23:07:31Z	ChristopherCastro	maintenance, REQ-NF, RFC
3909	Actualización Cities, LVL4	CLOSED	2017-05-19T08:34:35Z	ChristopherCastro	REQ-NF, resuelto, SQL-SCHEMA
3826	Emailable Behavior	CLOSED	2017-05-02T06:59:37Z	ChristopherCastro	refactorizar, REQ-F, RFC, SQL-SCHEMA, user story
3784	"Responsables" de expedientes	OPEN	2017-04-26T08:54:16Z	ChristopherCastro	prioridad alta, refactorizar, REQ-F, RFC, SQL-SCHEMA, user story, ☠ abandonado
3595	Sysadmin Dashboard	CLOSED	2017-03-29T12:08:00Z	ChristopherCastro	prioridad media, REQ-F, RFC, user story
3192	CRUD Emails	CLOSED	2017-02-15T11:56:54Z	Herberos	prioridad alta, REQ-F, RFC, user story
3182	Refactorización área financiero (Facturas) [DRAFT]	OPEN	2017-02-14T09:58:39Z	Herberos	refactorizar, REQ-NF, SQL-SCHEMA, ☠ abandonado
3136	Normalización Bancos [DRAFT]	OPEN	2017-02-08T11:00:28Z	ChristopherCastro	refactorizar, REQ-NF, RFC, SQL-SCHEMA, user story, ☠ abandonado
3134	Clasificador de Ficheros	CLOSED	2017-02-08T07:43:02Z	ChristopherCastro	REQ-F, RFC, user story
3109	Purga de Emails (GMAIL)	CLOSED	2017-02-06T12:20:54Z	ChristopherCastro	REQ-NF, resuelto, user story
3011	Normalización de Productos	CLOSED	2017-01-27T08:04:57Z	ChristopherCastro	prioridad alta, refactorizar, REQ-NF, resuelto, RFC, SQL-SCHEMA
2747	Clients API services	CLOSED	2016-12-27T12:51:39Z	ChristopherCastro	prioridad alta, refactorizar, REQ-F, resuelto, SQL-SCHEMA, user story
2743	e-mails Clientes	CLOSED	2016-12-26T15:00:46Z	ChristopherCastro	prioridad alta, resuelto
2662	Notificación de respuesta a email	CLOSED	2016-12-19T18:04:02Z	ChristopherCastro	bug, refactorizar, resuelto
2651	Lector actividad personas en "stand-by"	OPEN	2016-12-16T12:21:13Z	ChristopherCastro	REQ-F, SQL-SCHEMA, user story, ☠ abandonado
2554	Rollback facturas antiguas/mandamientos	CLOSED	2016-12-05T15:24:58Z	ChristopherCastro	bug, prioridad alta, REQ-NF, resuelto
2475	User Notification API	CLOSED	2016-11-28T10:06:52Z	ChristopherCastro	REQ-F, resuelto, RFC, SQL-SCHEMA, user story
2440	Emails - BCC, destinatarios adicionales y más	CLOSED	2016-11-23T17:08:06Z	ChristopherCastro	prioridad media, refactorizar, REQ-F, resuelto, RFC, user story
2428	Fix Google Script - Gmail integration	CLOSED	2016-11-23T11:32:25Z	ChristopherCastro	bug, REQ-NF, resuelto, RFC
2427	Refactorizar API "Documents"	OPEN	2016-11-23T10:19:33Z	Herberos	refactorizar, REQ-F, SQL-SCHEMA, user story, ☠ abandonado
2239	Refactor Call Provider	CLOSED	2016-11-04T16:21:28Z	ChristopherCastro	prioridad baja, refactorizar, REQ-F, resuelto

ANEXO

Visión Proyecto CRM

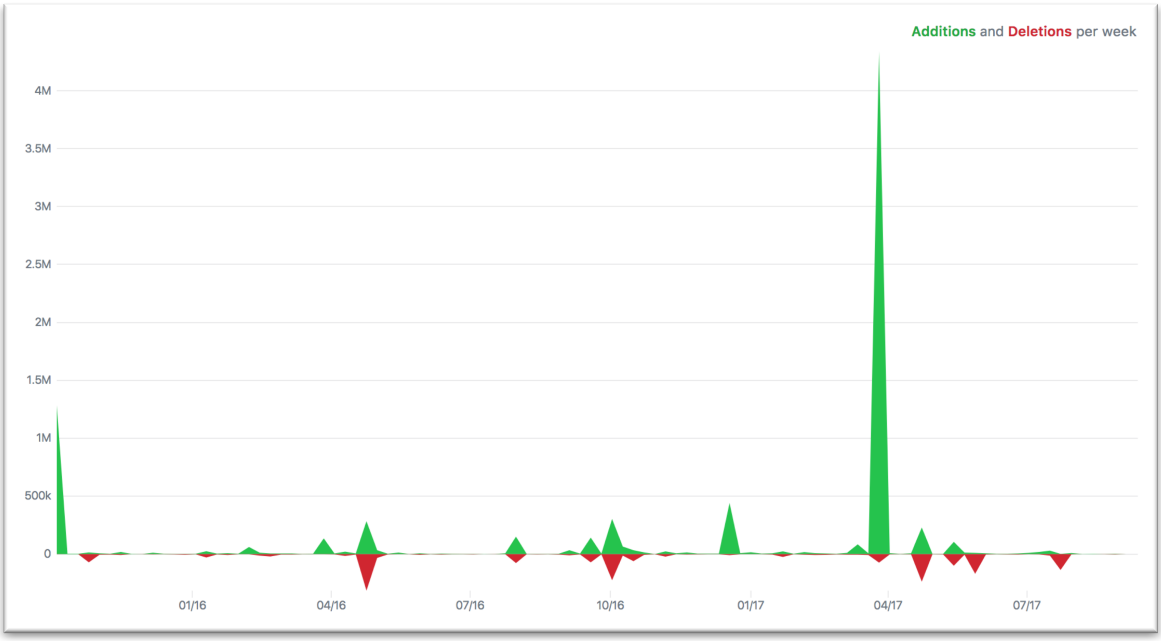
Commits

Número total de commits.



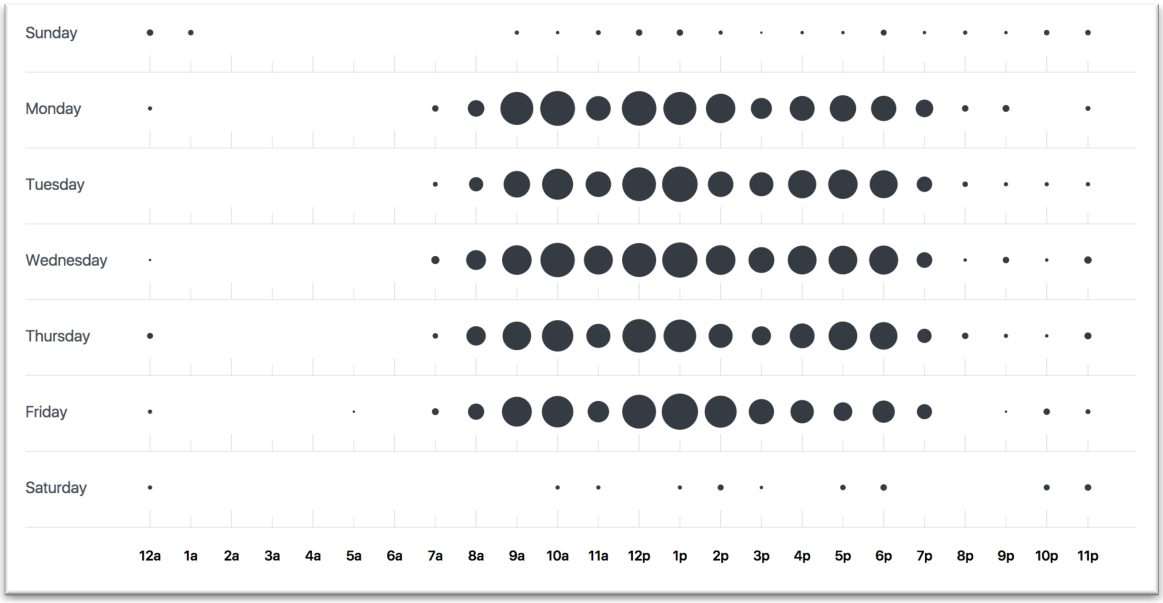
Code Frequency

Relación entre adición de nuevas líneas de código y líneas eliminadas.



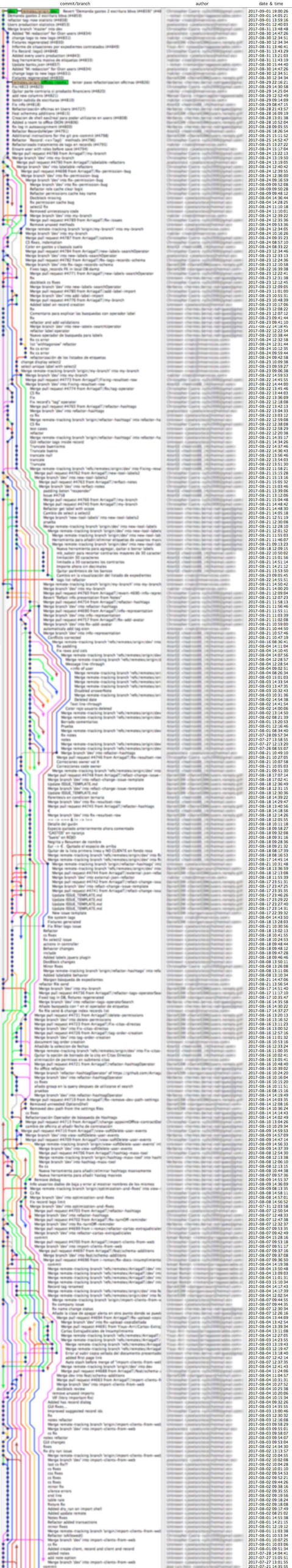
Punch Card

Estimación de productividad, días y horas de la semana.



ANEXO

Grafo Ramas GIT


commit/branch
author
date & time

ANEXO

Minería de Procesos

Minería de Procesos

Versión 0.7.1

Christopher Castro Alvarado

18 de junio de 2017

Índice general

Resumen

La minería de procesos es una disciplina que tiene como objetivo el descubrimiento, monitorización y mejora de los procesos de negocio mediante el análisis de registros de eventos ligados a los procesos y que se encuentran almacenados en los sistemas de información. En este pequeño estudio se presenta, por una parte, la aplicación de la minería en un entorno empresarial, y más concretamente al análisis de aquellos producidos por una bitácora de eventos CRUD. Y por otro, ligado con lo anterior, se propondrá una estrategia que permita el descubrimiento y composición de eventos, siempre y cuando el registro de evento posea determinadas características que así lo permitan.

Palabras clave: minería de procesos, análisis de procesos, big data, redes de Petri, bases de datos, mysql, eventos, logs, procesos, correlación de eventos, oracle, redolog, crud.

Índice de figuras

Índice de cuadros

CAPÍTULO 1

Introducción

En la actualidad, los sistemas de información de las organizaciones que brindan soporte a sus procesos de negocio registran información sobre "quién", "qué" y "cuándo" asociada a cada ejecución del proceso (instancia). Esta información puede ser aprovechada con técnicas de la minería de datos, más específicamente, de la minería de procesos para descubrir la realidad de cómo se están ejecutando los procesos y de esta forma tomar decisiones para mejorarlos *[VanAalst2011]*.

Las herramientas y algoritmos para la minería de procesos tienen su origen por el año 2004 en un pequeño grupo de investigación en una academia, y cuyos elementos han ido de forma gradual incorporándose en la industria. Últimamente los esfuerzos de los investigadores se centran, principalmente, en probar las herramientas y algoritmos en entornos reales con el fin de intentar detectar su limitaciones y problemas. Es por ello algunos autores enfatizan sobre la necesidad de contar con un mayor número de casos prácticos para probar los beneficios reales de la minería de procesos en entornos reales.

Las organizaciones, complejas y variables en su modo de operar, conducen hacia un creciente aumento en la aplicación de la gestión por procesos. En entornos altamente informatizados, se cuenta con un amplio abanico de técnicas de apoyo a la gestión por procesos. Minería de proceso, es una técnica de gestión de procesos que permite analizar los procesos de negocio basándose en trazas de comportamiento real registrados por un sistema de información. Las aplicaciones de estas técnicas permiten a las organizaciones identificar tendencias, descubrir patrones, detectar cuellos de botella, contrastar un modelo de proceso existen o incluso extraer uno a partir de actividad registrada por los sistemas de información.

En este documento se expondrá, por un lado, una problemática frecuente en proyectos de minería de procesos que cumplen determinadas características; se ofrecerá un modelado teórico de dicha problemática y se propondrá una solución que permite abordar el proyecto de una manera sencilla. Por otro lado, se aplicará la minería de procesos sobre un entorno empresarial real ligado al ámbito de la abogacía, aplicando por supuestos las ideas y conceptos aquí vistos. El empleo de este tipo de herramientas informáticas en el mundo de la abogacía son un aporte novedoso al sector, en estos entornos los procesos, por lo general, no están informatizados completamente y en muchos casos vagamente definidos o incluso inexistentes.

CAPÍTULO 2

Antecedentes

El objetivo de la minería de procesos es descubrir, monitorizar, y ofrecer procesos reales mediante la extracción de conocimiento de los eventos o actividad. Durante la última década, las técnicas de minería de procesos han madurado de forma considerable y están en la actualidad siendo integrados en diversos paquetes software comerciales *[Aalst2007]*.

Este "registro de eventos" (event log), como el que se puede observar en la Tabla 2.1, corresponde a trazas de ejecución de procesos de negocio, en donde se hallan las instancias o casos del proceso (ej. Solicitud de compra 1451), las actividades del proceso (ej. Envío de orden de compra al proveedor), las personas que ejecutan cada actividad (ej. Daniel García - Comercial), el inicio y fin de cada actividad e información complementaria inherente al propio caso. Para secciones posteriores es imprescindible aclarar ciertas definiciones básicas sobre qué partes conforman un *eventlog*: una fila por sí sola no representa una instancia completa de proceso, sino solo un evento. De ahí el nombre de "registro de eventos".

- Cada evento (cada fila) corresponde con una **actividad** que fue realizada dentro del proceso.
- Múltiples eventos son agrupados bajo una misma **instancia de proceso o caso**.
- Luego, cada caso o instancia genera una secuencia de eventos ordenadas en el tiempo.

Estos registros de eventos pueden encontrarse en sistemas de información de diverso tipo, por

case identifier	activity identifier	on	performer
case 3	activity A	today	John
case 2	activity A	yesterday	Brick
case 3	activity A	2 days ago	Sue
case 7	activity B	5 days ago	Drew
case 1	activity B	1 days ago	Mike
case 1	activity C	2 days ago	John
case 2	activity C	yesterday	Brick
case 4	activity A	yesterday	Sue
case 2	activity B	yesterday	John
case 2	activity D	yesterday	Eric
case 6	activity A	yesterday	Sue
case 4	activity C	yesterday	Drew
case 1	activity D	yesterday	Peter
case 3	activity C	yesterday	Sue
case 3	activity D	14:00	Eric
case 1	activity B	14:00	Drew
case 5	activity E	12:00	Clare
case 7	activity D	07:34	Clare
case 4	activity D	just now	Eric

Tabla 2.1: Ejemplo de un event log

ejemplo en los sistemas PAIS[1], sistemas de workflow, BPMS[2], ERP, CRM[3], entre otros. Todos estos datos son susceptibles de ser analizados con técnicas de minería de procesos para hacer explícita nueva información y de este modo extraer conocimiento que las organizaciones pueden luego utilizar para comprender su propio funcionamiento o mejorar sus procesos de negocio.

Las técnicas de minería de procesos requieren como entrada un registro de eventos "plano", es decir, debe ser claro, legible y por sobre todo bien estructurado. En 2.1 se muestra un ejemplo de un registro de eventos que las técnicas de minería de procesos esperan, en el registro se deben poder identificar de cada instancia, al menos, el caso, la actividad y el instante de tiempo en que se realiza o finaliza la actividad, a continuación, un descripción de cada uno de ellos:

- Caso: cada evento debe referenciar a un caso (por ejemplo, un expediente). Si un mismo evento resulta ser relevante para varios casos, entonces dicho evento debería ser replicado al momento de crear el registro de eventos.
- Actividad: cada evento debe corresponderse con una actividad. Los eventos referencian siempre a instancias de actividad, por ejemplo, ocurrencias de actividades en el modelo de procesos de actual.
- Marca Temporal: todos los eventos referidos a un mismo caso deberían estar ordenados. Es más, las marcas temporales no son solo necesarias para el ordenamiento temporal: sino que también resultan de utilidad a la hora de realizar estimaciones de rendimiento.

Junto con estos atributos esenciales, a veces se puede encontrar información complementaria en forma de atributos opcionales, por ejemplo:

- Recurso: Persona, máquina o componente que ejecuta el evento.

[1] Del inglés: Process Aware Information Systems
[2] Del inglés: Business Process Management
[3] Del inglés: Customer Relationship Management

- Cliente: Persona u organización para la cual el evento es ejecutado.
- Etc.

A continuación de exponen una serie de aplicaciones prácticas de la minería de procesos de interés para las organizaciones.

2.1 Descubrir el Modelo de Ejecución Real del Proceso

El descubrimiento de procesos abarca el conjunto de técnicas para la obtención de un modelo de procesos partiendo de una colección de eventos. Resulta interesante para algunas organizaciones la capacidad de descubrimiento de estos algoritmos, pues son capaces de identificar procesos reales partiendo únicamente de los registros de eventos asociados a sus ejecución. *[AalstAdriansyah2011]*.

Los modelos descubiertos por estos algoritmos pueden presentar la información de la ejecución del proceso desde diferentes enfoques:

- **Control de flujo:** Enfocada en el orden de ejecución de las actividades. Los procesos pueden representarse en algunos de los formalismos conocidos para el modelado de procesos de negocio: Redes de Petri (algoritmo Alpha Miner *[Aalst2004]*), Redes de Flujo de Trabajo (WFN siglas en inglés), Redes Heurísticas (algoritmo Heuristics Miner *[WeijtersAalst2003]*) en dependencia del algoritmo de descubrimiento empleado y la expresividad de la notación en cuestión.
- **Perspectiva organizacional:** Identificando las responsabilidades y relaciones de los recursos involucrados en la ejecución de los procesos. Partiendo de las redes sociales descubiertas en la organización pueden identificarse relaciones entre los recursos humanos de asignación de trabajo, sub-contratación o delegación, asignación de tareas, trabajo en equipo, personal con tareas similares *[AalstReijersSong2005]*. Adicionalmente es posible realizar análisis de los roles y equipos de trabajo presentes en la organización proporcionando información sobre qué recursos humanos intervienen en la ejecución de las actividades del proceso.
- **Datos:** Mostrando información sobre el flujo de los datos durante la ejecución del proceso minado. *[RozinatAalst2006]*

2.2 Mejorar Modelos de Procesos Existentes

La mejora o extensión de procesos se refiere a aquellas técnicas para analizar el rendimiento de un proceso y/o extender su modelo, con el objetivo de que se refleje correctamente la ejecución real del proceso al cual representa.

Para su análisis utiliza un modelo de proceso y el registro de eventos asociado a la ejecución del mismo. Permite detectar elementos relevantes para la optimización de los procesos tales como: cuellos de botella, tiempos de procesamiento, frecuencias de ejecución de las actividades, caminos del proceso que nunca son recorridos, etc. *[Hornix2007]*

2.3 Descubrir cuellos de Botella

Mediante la representación animada de flujos de casos reales del proceso es posible localizar (como se aprecia en Figura 2.1) cuellos de botella y actuar en consecuencia para mejorar el desempeño general del proceso. Esto es posible siempre y cuando las trazas de eventos posean una marca temporal, de al menos, el inicio de la actividad.

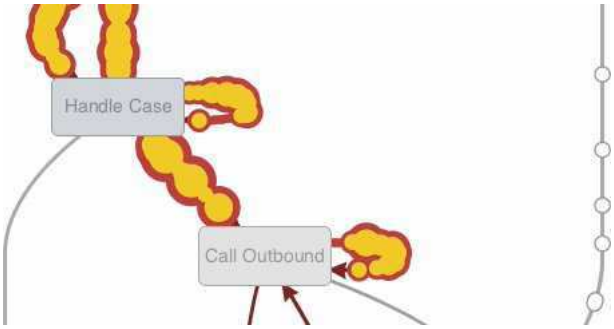


Figura 2.1: Descubrimiento de cuellos de botella con *Disco*.
Captura de pantalla de software *Disco* en donde se aprecia una gran concentración de eventos entre dos actividades; a la derecha se pueden observar trazas que "circulan" de forma holgada, sin solapamientos temporales, entre dos actividades.

2.4 Descubrir Relaciones Entre Departamentos y el Personal

Con la aplicación de técnicas de minería de datos puede construir la red social del proceso (social network) para analizar la interacción entre los individuos y descubrir bucles (loops) que pueden demorar la ejecución de un proceso. Para esto, evidentemente es indispensable que en cada evento se identifique de alguna manera al *Originator*, es decir, la persona física que realiza la actividad registrada en el evento. *[AalstReijersSong2005]*

En definitiva, la minería de procesos busca la mejora de los procesos a través de la aplicación de herramientas de análisis de datos. En Figura 2.3 se ofrece de forma esquematizada el flujo de trabajo de la minería de datos: descubrir (*Discovery*), Verificación (*Conformance*) y Mejora (*Enhacement*).

2.5 Tipos de Proyectos de Minería de Procesos

Los proyectos de minería de procesos pueden ser clasificados en básicamente tres, esto según *[VanAalst2011]*:

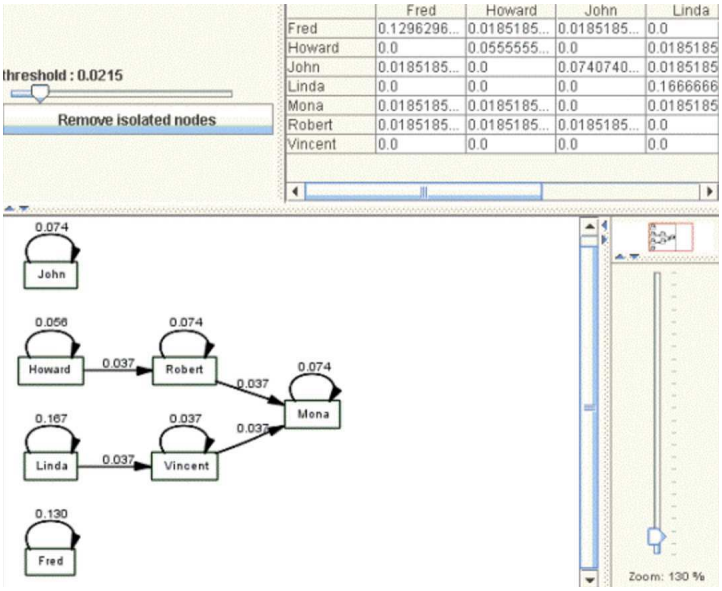


Figura 2.2: Ejemplo de red social con *ProM 6*.
Captura de pantalla de software *ProM 6*. Ejemplo de resultado. La red social generada se muestra en forma de matriz (arriba a la derecha) y en forma de grafo (abajo).

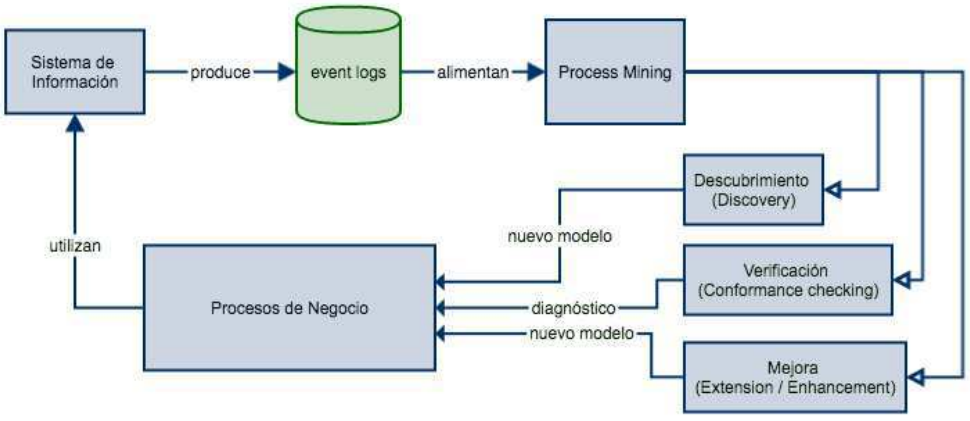


Figura 2.3: Flujo de trabajo en la minería de procesos.
Esquematización del flujo de trabajo de la minería de procesos.

- Basados en datos (*data-driven*): donde no existen objetivos establecidos, son de carácter exploratorio (por lo que también se les denomina "guiados por la curiosidad") y se intentan descubrir nuevos aspectos importantes respecto a la ejecución real de los procesos.
- Basados en preguntas (*question-driven*): donde se cuenta con una serie de hipótesis a las cuales se desea ofrecer respuesta, verificar o rechazar en base a información inferida de la minería de procesos.
- Basados en objetivos (*goal-driven*): donde se establecen como objetivos la mejora de los indicadores clave de desempeño de los procesos.

Los proyectos guiados por los datos carecen de una fase inicial de definición, puesto que se trata de simplemente explorar o "curiosear" se parte un conjunto de datos que se han de analizar para detectar patrones y elementos de utilidad que puedan ser utilizados para mejorar o diagnosticar el proceso. Por otro lado, los proyectos basados en objetivos hacen especial hincapié en mejorar el desempeño de los indicadores de gestión, por lo que se debe realizar un análisis centrado en detectar incongruencias o "gaps" entre indicadores; a diferencia de los proyectos basados en preguntas, donde este paso no es aplicable.

2.6 Calidad de los Datos

La aplicación de la minería de procesos requiere de un enfoque integrador entre la gestión de procesos de negocio y la tecnología. Resulta de suma importancia la colaboración y trabajo conjunto entre profesionales de distintas disciplinas, como la informática, jurídica, contabilidad, etc.

La fuente de toda técnica de minería de proceso son los registros de eventos de los sistemas de información. Estos registros de eventos pueden ser clasificados en cinco niveles teniendo en cuenta su calidad.

- Nivel 5: Nivel más alto, el registro de eventos es de excelente calidad (confiable y completo) y los eventos están bien definidos. Los eventos se registran de manera automática, sistemática, confiable, y segura. Se toman en cuenta adecuadamente consideraciones acerca de la privacidad y la seguridad. Además, los eventos registrados tienen una semántica clara.
- Nivel 4: Los eventos se registran automáticamente y de manera sistemática y confiable. A diferencia de los sistemas operando a nivel 3, se da soporte de manera explícita a nociones tales como instancia de proceso (caso) y actividad.
- Nivel 3: Los eventos se registran automáticamente, pero no se sigue un enfoque sistemático para registrar los eventos. Sin embargo, a diferencia de los registros de eventos en el nivel 2, hay algún nivel de garantía que los eventos registrados se ajustan con la realidad (el registro de eventos es confiable pero no necesariamente completo). Aunque se necesita extraer los eventos de una variedad de tablas, se puede asumir que la información es correcta.
- Nivel 2: Los eventos se registran automáticamente, como un subproducto de algún sistema de información. La cobertura varía, no se sigue un enfoque sistemático para decidir qué

eventos se registran. Además, es posible pasar por alto el sistema de información. Por lo tanto, podrían faltar eventos o estos podrían no registrarse correctamente.

- Nivel 1: Los registros de eventos son de mala calidad. Los eventos registrados podrían no corresponder a la realidad y podrían faltar eventos. Los registros de eventos en los cuales los eventos se registran manualmente suelen tener dichas características. Ejemplo: trazas dejadas en documentos en papel que se trasladan a través de la organización.

Las técnicas de minería de procesos pueden ser aplicadas sobre registros de eventos de nivel 5, nivel 4 y nivel 3. Aunque también es posible aplicarlos a niveles inferiores, por lo general su análisis resulta tremendamente complicado y sus resultados distan de ser confiables. Por ejemplo, no tiene absolutamente ningún sentido aplicar estas técnicas a registros de nivel 1; registros que no reflejan la realidad.

CAPÍTULO 3

Descripción del Problema

La mayoría de los sistemas de información no registran eventos de forma que puedan ser directamente utilizados. Únicamente sistemas orientados a procesos registran eventos "planos" como los que se muestran en la Tabla 2.1. Por lo cual, para generar un registro de eventos apto para la minería de procesos, es necesario muchas veces recolectar datos de multitud de fuentes, como puede ser un simple fichero de texto, una hoja Excel, un fichero de logs o una tabla de una base de datos. Sin embargo, no se debe esperar en ningún caso que todos los datos estén bien estructurados y provengan de una misma fuente. La realidad es que los datos de los eventos están generalmente fragmentados y repartidos entre distintas fuentes de datos, por lo que se necesitan esfuerzos adicionales para recolectar información relevante. *[VanAalst2016]*

Es más, casi para la totalidad de los proyectos de minería de procesos los eventos deben ser extraídos de tablas de bases de datos convencionales de una manera completamente ad- hoc, a pesar de que existen herramientas especializadas que facilitan la tarea, al final siempre se terminan realizando simples consultas a la base de datos para extraer eventos e ir construyendo un registro válido.

El principal problema al intentar extraer eventos a partir de tablas es que éstas solo proporcional una visión actual del sistema, lo que dificulta o imposibilita la reconstrucción de trazas de eventos. Por ejemplo, en un sistema donde se marca la fecha de la última modificación de una tupla sería imposible reconstruir un histórico para esta tupla (caso), pues no se tiene información sobre todas las modificaciones anteriores ni información sobre qué se ha modificado exactamente. Otra problemática son las eliminaciones de tuplas, que por lo general son imposibles de recuperar a no ser que se apliquen técnicas como *softdelete*.

Este documento se centra en un escenario concreto, en donde se dispone de un registro de cambios de una base de datos, es decir, aquel que registra cuando una tupla es insertada, modificada o eliminada. Dicho registro puede ser interpretado como un registro de eventos en principio apto para la aplicación de técnicas de minería de procesos. El problema con este tipo de registros es la forma en que los eventos son almacenados, pues existen eventos que pueden no estar directamente relacionados con un caso en concreto, sino que en cierto modo un evento puede pertenecer a la traza de otro caso que únicamente existe de manera implícita.

CAPÍTULO 4

Solución Propuesta

Si bien en el capítulo anterior se exponen problemáticas sobre un escenario concreto, donde existe una base de datos y un registro de cambios del mismo. En este capítulo se abordarán estos problemas de una manera generalizada. La idea es proporcionar unas bases conceptuales que permitan una extensión o especialización sobre el escenario que se plantea como problema.

En concreto, se parte de la suposición que se cuenta por un lado con un registro de cambios de una base de datos y por otro un modelo relacional que describa la relaciones entre tablas. La idea es que, combinando ambos elementos, es posible entonces definir (o mejor dicho, descubrir) en términos abstractos una serie de nuevos eventos que existen de manera implícita en el propio registro.

4.1 Grafos, Eventos y Correlaciones

En este apartado se intentará generalizar una solución a la problemática expuesta en secciones anteriores. Se parte de suposición de que se cuenta con dos elementos imprescindibles: un registro de eventos atómico y un grafo de correlación de tipos, a continuación, se detallan las características de estos elementos y se explicará por qué resultan necesarios.

El **registro de eventos atómico**: aquel que registra las operaciones CRUD de cada “caso”, es decir cada vez que un caso es creado, leído, modificado o eliminado genera un evento que refleja dicha **actividad** en el tiempo, como se observa en la tabla 4.1. Luego, un **evento atómico** es aquel

event	case	case type	activity	on
1	case 3	user	create	today
5	case 2	order	update	yesterday
9	case 3	user	delete	2 days ago
10	case 7	order	create	5 days ago
11	case 1	user	update	1 days ago
90	case 1	user	retrieve	2 days ago
101	case 2	order	create	yesterday
3	case 4	order	create	yesterday
4	case 2	order	retrieve	yesterday
12	case 2	order	retrieve	yesterday
77	case 6	payment	delete	yesterday
23	case 4	order	update	yesterday
21	case 1	user	create	yesterday
98	case 3	user	retrieve	yesterday
501	case 3	user	update	14:00
24	case 1	user	update	14:00
90	case 5	payment	update	12:00
33	case 7	order	retrieve	07:34
41	case 4	payment	retrieve	just now

Tabla 4.1: Ejemplo de un registro de eventos atómico

cuya actividad es una actividad CRUD, por lo que (abusando del lenguaje) es posible referirse a ellos como “evento de creación”, “evento de modificación”, etc.

Por lo general, en un registro de eventos atómicos no debería haber casos con más de un evento de creación o eliminación, aunque es admisible y podría ocurrir en sistemas en los que se hacen uso de técnicas como eliminaciones débiles (*softdelete*), por lo que no existen restricciones en este sentido. Las únicas dos restricciones adicionales que ha de cumplir un registro de eventos atómicos son:

1. Ha de reflejar de alguna forma el **tipo de caso**, en ocasiones esto queda plasmado de forma implícita en el propio identificador de los casos. El tipo de caso puede entenderse como el proceso al que pertenecen los distintos casos. En última instancia determinan los distintos modelos de procesos que es posible obtener de un mismo registro de eventos.
2. Se ha de poder identificar de manera inequívoca un evento en concreto, es decir, cada entrada del registro debe poseer un identificador.

Por otro lado, al aplicar técnicas de minería de procesos, generalmente se realiza una purga selectiva de los eventos registrados en función del modelo que se pretenda obtener. Por ejemplo, si interesa conocer el proceso de compras de una organización lo lógico es centrarse en aquellos eventos asociados casos que guarden relación con el foco de estudio. Como bien se ha comentado en secciones anteriores, un evento viene caracterizado por tres propiedades obligatorias: identificador de caso, actividad y una fecha.

El identificador de caso, también denominado *identificador de instancia de proceso*, es necesario

para distinguir entre diferentes ejecuciones de un mismo proceso, por lo que dicho identificador depende directamente del dominio del proceso. Por ejemplo, en un departamento que gestiona las órdenes de compra, el identificador de caso podría ser el identificador de la propia orden; en un proceso de órdenes de compra, el proceso de gestionar una orden en concreto representa una instancia de proceso. Por ello es necesario que, para cada evento se conozca el caso al que referencia, así las técnicas de minería de procesos pueden comparar diferentes ejecuciones de un mismo proceso unas con otras.

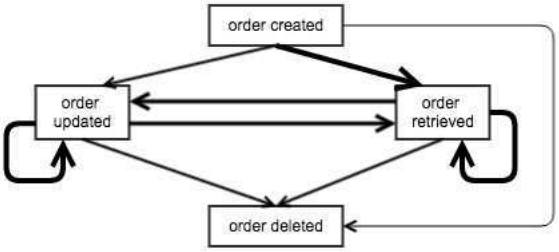


Figura 4.1: Ejemplo de un modelo de procesos obtenido directamente de un registro de eventos atómico sobre los tipos de casos "order".

El problema ocurre cuando existen dependencias o relaciones entre casos fuera del alcance del proceso a estudiar. Situación habitual en entorno donde el registro de eventos con el que se cuenta es de tipo atómico, como por ejemplo:

- Los **Redo Log** de Oracle, que reflejan las actividad de creación (INSERT), modificación (UPDATE) y eliminación (DELETE) de cada tabla de la base de datos.
- Los sistema gestores de contenidos (CMS), suelen proporcionar sistemas de versionado y control de cambios sobre los contenidos publicados.
- Sistema que empleen un registro a modo de "histórico".

Continuando con el ejemplo de órdenes de compra anterior, y considerando que el único registro de eventos con el que se cuenta es de tipo atómico, entonces el único modelo de procesos que es posible construir para la gestión de órdenes de compra es aquel en donde, únicamente, se ven reflejadas las **actividades** de: creación, modificación, lectura y eliminación de una orden. En definitiva, un modelo que no refleja el comportamiento real del proceso, como el que se presenta en Figura 4.1. Por ejemplo, el pago de una orden de compra puede haber ocurrido después de haber sido creada la orden, pero dicho evento de creación del pago queda ligado al propio caso del pago, es decir, pertenecen a tipos de casos distintos.

Sin embargo, es posible generar o construir nuevos eventos como la composición de eventos atómicos que aportan mayor información al modelo de proceso obtenido, para ello es necesaria una descripción formal de las dependencias entre los tipos de casos en el registro de eventos. En el ejemplo anterior, los tipos de casos "Orden de compra" y "Pago" están relacionados de cierta forma. De este modo, y continuando con el mismo ejemplo, cada vez que un pago es registrado (creado), esta información quizás resulte de interés al estudiar el proceso de órdenes de compra. Para reflejar esta relación entre dos tipos de casos es necesario una estructura que

permita describir esta "conexión". Por ello, el segundo elemento imprescindible es un grafo G que describa las relaciones entre cada tipo de caso, denominaremos a este elemento *grafo de correlación de tipos de casos*, o para abreviar (y abusando del lenguaje), simplemente **Grafo de Correlaciones**.

En este grafo los nodos representan tipos de caso, y cada arco indica si dos tipos de casos están relacionados entre sí. Más adelante se entrará en mayor detalle sobre qué significa que dos tipos de casos estén relacionados. Por último, es importante señalar que en el contexto actual los términos "tipo de caso" y "nodo" se utilizarán de forma intercambiable.

Una vez se cuenta con un registro de eventos atómico y un grafo que refleje las relaciones entre cada tipo de caso, se introduce el concepto de **correlación de eventos**:

4.1.1 Correlación de Eventos

En el ámbito de la minería de procesos se entiende como *correlación de eventos* al proceso de encontrar relaciones entre eventos que pertenecen a la misma instancia de ejecución en un modelo de procesos. La correlación de eventos es una tarea de difícil solución y que en la práctica resulta (en la mayoría de casos) imposible de generalizar, pues existen multitud de formas o criterios a aplicar a la hora de determinar cuándo dos eventos están correlacionados o no, y en muchos casos, dichos criterios resultan totalmente subjetivos. Explorar todas las posibles correlaciones resulta computacionalmente inviable, y además, en la práctica solo unas pocas correlaciones resultan de interés. En este estudio se ofrece una visión general de un caso particular de correlación basado en grafos de correlaciones. Idea que, como se verá más adelante, puede ser extendida y especializada a una variedad de escenarios que cumplan con los dos elementos imprescindibles mencionados en párrafos anteriores. Para ello es necesario introducir el concepto de *condición de correlación*. *[HamidReza2009]*

4.1.2 Condición de Correlación

Una condición de correlación, denotado por $\psi(a, b)$, de forma generalizada es un predicado binario definido sobre dos elementos, el predicado es verdadero en caso de que *a* y *b* están correlacionados y falso cuando no lo están.

El criterio o condición depende totalmente del dominio en que dicha condición es definida. Por ejemplo, si tanto *a* como *b* son elementos comparables entre sí, entonces una condición de la forma $a.x = b.x$ indica que ambos elementos quedan correlacionados a través de la igualdad entre los valores de un mismo atributo común. Una condición de este estilo se denomina *condición atómica* o *regla atómica*. Una condición conjuntiva (y disyuntiva, respectivamente) consiste en una conjunción (y disyunción respectivamente) de condiciones atómicas. Dicho esto, es posible definir instancia de proceso de la siguiente manera:

Instancia de Proceso, Una instancia de proceso pi es una secuencia de eventos $\{v_1, v_2, \cdots, v_k\}$ (un subconjunto de eventos de un registro de eventos $\mathcal{L}$), tal que para todo evento $v_i \in pi$ existe al menos un evento $v_j \in pi : i \neq j$. tal que v_i está directamente correlacionado con v_j, es decir, $\psi(v_i, v_j) \rightarrow verdadero$ *[HamidReza2009]*.

4.1.3 Correlación y Composición: Eventos y Tipos

Una vez introducida la noción de correlación entre dos elementos, y partiendo del supuesto que se cuenta con un registro de eventos atómico y con un grafo que describa las correlaciones entre los distintos tipos de casos, entonces es posible definir la composición de tipos de eventos de mayor nivel, eventos que aportan una mayor información al modelo de procesos de un tipo de caso en particular.

Es importante recalcar la diferencia entre **eventos** y **tipos de eventos** (similar a como ocurre con **caso** y **tipo de caso**), el primero hace referencia a una entrada en un registro de eventos, mientras que el segundo hace referencia a un conjunto de eventos que comparten una determinada característica. Por otra parte, en los **casos** y **tipos de casos** ocurre de igual forma. En la minería de procesos, un proceso está conformado por casos, a su vez cada caso está formado por eventos tales que están relacionados precisamente con un caso. El tipo de caso, nos permite agrupar distintos casos bajo un mismo nombre, se asume que todos los casos de un mismo tipo poseen características comunes.

Dicho esto, a continuación, se introducen algunas definiciones necesarias para conseguir definir la construcción de tipos de eventos:

- $\mathcal{L}_a$ un registro de eventos atómico.
- CT el conjunto de todos los tipos de casos existentes en $\mathcal{L}_a$
- $G = (N, E)$ un grafo que describe las relaciones entre distintos tipos de casos existentes en $\mathcal{L}_a$.
  - N el conjunto de nodos del grafo *G*
  - E el conjunto de arcos del grafo *G*
- $N_\psi(n_1, n_2) : n_1, n_2 \in N$ predicado de correlación sobre nodos. Describe la correlación entre dos nodos de G, en este sentido el predicado es verdadero si y solo si existe un arco que conecte ambos nodos. Se considera que un nodo está siempre conectado consigo mismo, es decir $N_\psi(n_1, n_1) \rightarrow verdadero$.

Sea $n \in N$ una **tipo de caso**, es decir, un nodo del grafo. Por ejemplo, *order* representa a todos los casos relacionados con órdenes de compra. Entonces:

- n^+ representa a todos los eventos atómicos de creación para un tipo de caso *n*. Por ejemplo, de la tabla 4.1: $order^+ = \{10, 101, 3\}$
- n^- representa a todos los eventos atómicos de eliminación para un tipo de caso *n*. or ejemplo, de la tabla 4.1: $order^- = \emptyset$
- n^* representa a todos los eventos atómicos de modificación para un tipo de caso *n*. Por ejemplo, de la tabla 4.1: $order^* = \{5, 23\}$
- $n^!$ representa a todos los eventos atómicos de lectura para un tipo de caso *n*. Por ejemplo, de la tabla 4.1: $order^! = \{4, 12, 33\}$
- $\alpha(n)$ función que devuelve el conjunto de todos los **tipos de eventos atómicos** registrados en $\mathcal{L}_a$ para una tipología de caso (*n*). Por ejemplo, $\alpha(order) = \{order^+ order^*\}$.

- $A = \alpha(n_1) \cup \cdots \cup \alpha(n_i)$ el conjunto de todos los posibles tipos de eventos atómicos sobre $\mathcal{L}_a$, por ejemplo $A = \{order^+, order^-, user^+, user^*, \cdots\}$.
- $\beta(a) : A \to CT$ función que dado un tipo de evento atómico devuelve el tipo del caso asociado. Por ejemplo, $\beta(order^+) = order$.

Definimos a continuación el conjunto N_n de todos los posibles tipos de eventos correlacionados con un tipo de caso *n*:

$$\boxed{N_n = V \subseteq A : \forall v \in V, N_\psi(n, \beta(v)) \to verdadero} \tag{4.1}$$

Finalmente, un **tipo de evento compuesto**, es aquel que puede obtenerse como composición de tipos de eventos atómicos, siempre y cuando los nodos asociados a dichos tipos de eventos atómicos guarden cierta relación con un mismo **tipo de caso objetivo** *n*. Entonces es posible definir una **tipología de eventos compuestos** como:

$$\boxed{n^c = p \in \mathcal{P}(N_n) - \emptyset} \tag{4.2}$$

Es decir, un subconjunto **no vacío** de tipos de eventos atómicos tales que están correlacionados con el tipo de caso *n*.

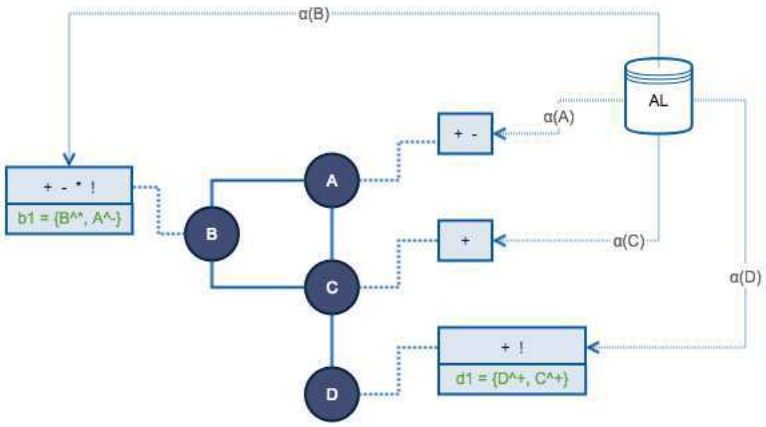


Figura 4.2: Esquematización de grafo y eventos compuestos.
Los nodos A-D representa tipos de casos, AL representa el registro de eventos atómico. Para cada tipo de caso se pueden observar por un lado los tipos de eventos atómicos existentes en AL, y por otro los eventos compuestos *b1* y *d1*.

4.1.4 Eventos e Instancias

Las definiciones anteriores establecen las bases para comprender tipos de eventos y un registro de eventos atómicos entre otras cosas. Básicamente describen, en términos abstractos, un determinado tipo de eventos que puede ser obtenidos como la agregación de otros tipos de eventos

más simples. Sin embargo, no se ofrece ningún mecanismo para relacionar **eventos** unos con otros para así construir trazas de ejecución. En la práctica, las técnicas de minería de procesos trabajan directamente sobre instancias de procesos, es decir, en secuencias de **eventos concretos** (trazas) que describen una ejecución del proceso, y no sobre una tipología de eventos como ocurre en las definiciones anteriores.

La correlación definida anteriormente permite únicamente identificar una tipología de eventos, sin llegar a relacionar eventos con instancias. Lo que en la práctica resulta insuficiente. De aplicar dichos conceptos directamente solo se obtendrían eventos compuestos sin un caso asociado. En definitiva, los conceptos anteriores sólo nos dicen que cierta combinación de determinados eventos referencian de algún modo a **algún** caso, cuya única información conocida es su "tipo de caso", no nos dicen cuál caso ni cuales eventos del registro en concreto.

Para ejemplificar y visualizar la situación, consideremos la tabla 4.1 y el grafo en Figura 4.3 que describe la relación entre los tipos de casos "order", "user" y "payment".

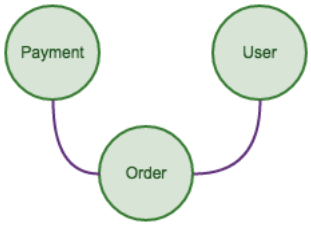


Figura 4.3: Ejemplo grafo de correlaciones para un proceso de órdenes de de compra.
Representa la relación existente entre los tipos de casos "Order", User" y "Payment".

El grafo en Figura 4.3 nos dice que los eventos asociados a tipos de casos "Order" guardan cierta relación con los de "Payment", y "Order" con los de "User". Las definiciones de la sección anterior nos dicen que entonces es posible generar un evento ligado a casos de "Order" tal que agrega información de eventos ya existentes en "Order" y/o en "Payment". Por ejemplo, el registro de un pago ($payment^+$) seguramente guarda relación con algún caso en "Order", pero tanto el pago en sí como la orden en concreto, es decir las instancias o casos, son desconocidos.

Todas las ideas y definiciones anteriores pueden ser extendidas de forma natural y aplicarse sobre eventos concretos de un registro de eventos. Para ello es necesario re- definir la función $\beta(v)$ y el predicado de correlación, de forma que actúen sobre eventos de un registro y no sobre tipos de eventos:

- Sea $v \in \mathcal{L}_a$ un evento atómico de un registro de eventos atómico,
- $\beta(v) : \mathcal{L}_a \to CT$ función que dado un evento atómico devuelve el tipo del caso asociado. Por ejemplo, del registro en la tabla 4.1, $\beta(v_5) = order$
- $V_\psi(v_i, v_j) : v_i, v_j \in \mathcal{L}_a$ predicado de correlación sobre eventos atómicos de un registro de eventos atómico. Verdadero si están correlacionados y falso cuando no lo están. La condición de correlación se estudiará a continuación.

En un enfoque tradicional, en la minería de procesos, los eventos quedan correlacionados cuando referencian a un mismo caso. Como se comentó en el apartado anterior, para ello se define un predicado de correlación sobre **atributos de eventos**, denotado como $\psi(v_x.a_i, v_y.a_j)$; predicado booleano sobre los atributos a_i y a_j de los respectivos eventos v_x y v_y. En un registro de eventos se asume que que el identificador de caso queda determinado por un atributo común para todos los eventos del registro, cada evento siempre posee un atributo que identifica al caso asociado, entonces el **predicado atómico** $v_x.caseID = v_y.caseID$ permite correlacionar todos los eventos pertenecientes a un mismo caso, siendo obviamente "caseID" el atributo cuyo valor referencia a los casos.

En definitiva el predicado de correlación sobre nodos de un grafo $N_\psi(n_1, n_2)$ resulta insuficiente a la hora de correlacionar eventos a trazas de ejecución (casos), aunque bien es cierto que éste ayuda a delimitar o aproximarse a la solución del problema. Dicha problemática a sido extensamente estudiada en *[HamidReza2009]*, ahí se ofrecen multitud de soluciones dependiendo del dominio del problema. Por ello en este documento no se aportan mayores desarrollos al respecto.

Como primer acercamiento, es posible definir el predicado de correlación sobre eventos en función de su tipología $V_\psi(v_x, v_y) = N_\psi(\beta(v_x), \beta(v_y))$, predicado que como bien se comentó anteriormente **resulta insuficiente**, es decir, falta algún elemento que permita relacionar los eventos con casos, es decir, instancias de procesos.

Técnicamente, lo que se necesita es un conjunto de instancias de procesos PI (casos) y algún mecanismo para relacionar eventos a esos casos, es decir, un predicado de correlación para eventos definido en función de instancias de procesos:

- Sea $pi \in PI$ una instancia de ejecución de un proceso.
- $P_\psi(v_1, v_2)$ predicado de correlación entre eventos, verdadero si dos eventos pertenecen a la misma instancia, es decir si $v_1, v_2 \in pi$

Luego es posible definir un predicado de correlación aplicando la conjunción de estas ideas, basada en nodos y basada en instancias:

$$\boxed{V^n_\psi(v_1, v_2) = N_\psi(n, \beta(v_1)) \wedge N_\psi(n, \beta(v_2)) \wedge P_\psi(v_1, v_2)} \tag{4.3}$$

El predicado es verdadero si ambos eventos pertenecen a tipos de casos correlacionados con un tipo de caso objetivo *n* y además pertenecen a la misma traza de ejecución. Falso en cualquier otro caso. Nótese que una condición no implica necesariamente a la otra; que dos eventos estén correlacionados con un tipo de caso *n* no implica que siempre pertenecen al mismo caso o instancia de proceso.

Es posible agregar información de distintos eventos atómicos para producir un nuevo evento compuesto para un tipo de caso *n* dado. Para ello definimos primero el conjunto V_n, formado por todos los eventos atómicos correlacionados respecto de *n* y con respecto un mismo caso:

$$\boxed{V_n = V \subseteq \mathcal{L}_a : \forall v_x \in V, \exists v_y \in V | x \neq y \wedge V^n_\psi(v_x, v_y) \rightarrow verdadero} \tag{4.4}$$

Luego el conjunto potencia de V_n, denominado $\mathcal{P}(V_n)$, es aquél formado por todos los subconjuntos de V_n. Finalmente es posible definir un **evento compuesto** respecto de *n* como la proyección de uno de estos subconjunto (no vacío) sobre un registro de eventos, es decir:

$$\boxed{v_n = \pi(p) : p \in \mathcal{P}(V_n) - \emptyset} \tag{4.5}$$

Este nuevo evento generado de forma artificial debe contar con, al menos, las tres propiedades obligatorias: identificador de caso, actividad y tiempo. De estas tres dos de ellas son conocidas, el identificador de caso y el nombre de la actividad que es definido en el proceso de composición a través de la condición de correlación. Sin embargo, el tiempo requiere de un tratamiento especial, pues este nuevo evento debe ocurrir en un único instante de tiempo, pero al tratarse de la proyección de un subconjunto de eventos se cuenta en realidad con tantos instantes de tiempo como elementos en el subconjunto. Es por ello que la función de proyección depende por completo del dominio del problema y el tipo de composición, por ejemplo, el menor de los tiempos, el más frecuente, el tiempo medio o incluso forzar composiciones de un único elemento son todas posibles soluciones. Aunque, **se recomienda escoger el instante de tiempo del evento que ocurre más tarde**, de este modo el nuevo evento generado ocurre en la práctica cuando ha "terminado" el último de sus eventos.

4.2 Extensión, Modelo Relacional

Los conceptos y definiciones anteriores pueden ser extendidas a cualquier escenario que cuente con un registro de eventos atómico y un grafo que describa las relaciones entre los distintos tipos de casos. En esta sección se estudiará su aplicación sobre un escenario típico, en donde el registro de eventos es en realidad un registro de cambios sobre una base de datos cuyos esquemas y relaciones son conocidos.

Bajo este escenario, el registro de cambios sobre cada una de las tablas puede ser interpretado como un registro de eventos atómico. Pero con la principal diferencia de que las operaciones de lectura no son registradas, pues no constituyen un cambio de las tablas. Es decir, contamos con un registro que plasma las operaciones de creación (INSERT), modificación (UPDATE) y eliminación (DELETE) de las tablas en una base de datos, cada entrada en este registro de cambios puede ser visto como un evento, además, se asume que cada una de estás entradas indican la tabla y la fila afectada, que pueden ser entendidas como tipo de caso y caso respectivamente. A continuación se ponen en claro estas equivalencias:

- Entrada en registro de cambios = evento
- Tabla afectada (información de cada entrada) = tipo de caso asociado al evento
- Fila de la tabla afectada (información de cada entrada) = caso

Por otro lado, un modelo relacional que describa las relaciones o dependencias entre las distintas tablas puede ser entendida como un grafo de correlaciones como el descrito en secciones anteriores, necesario para la definición de un predicado de correlación que permita la definición de tipos de eventos compuestos.

Para este estudio se utilizará como ejemplo el modelo relacional en Figura 4.4. Las distintas tablas del modelo pueden ser vistas como tipos de casos. Estas ideas pueden ser unidas para producir de forma trivial un grafo de correlaciones como el que se puede ver en Figura 4.5. En la figura se observan dos tipos de arcos. Aquellos con trazo continuo, deben ser interpretados como enlaces "fuertes" o "físicos" entre dos tablas. El segundo tipo de arco, trazo discontinuo, representa un enlace "débil" o "virtual" entre tablas, representan relaciones que no existen físicamente pero sí en términos semánticos. Recordemos que las definiciones de la sección anterior

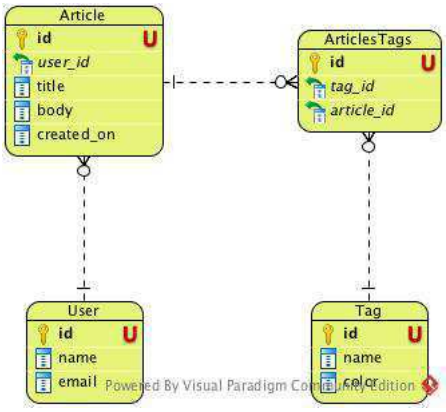


Figura 4.4: Modelo relacional de ejemplo, blog.
Un usuario puede escribir muchas entradas de un blog (Article), a su vez una entrada puede ser etiquetada con múltiples etiquetas (Tag) a la vez, una misma etiqueta puede haber sido utilizada para etiquetar a más de una entrada.

son agnósticas en este sentido, pues solo se exige que el grafo represente las correlaciones entre los tipos de casos, de este modo *Artículos* y *Etiquetas* en efecto poseen cierta relación en términos semánticos: "una entrada puede ser etiquetada con múltiples etiquetas (Tag) a la vez, una misma etiqueta puede haber sido utilizada para etiquetar a más de una entrada"

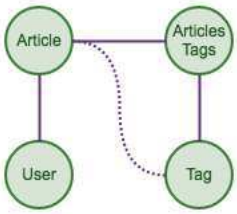


Figura 4.5: Grafo de correlaciones del modelo relacional.
Cada tabla representa un tipo de caso, los arcos no reflejan la cardinalidad sino que únicamente indican que existe una relación entre dos tipos de casos. Las relaciones *n:m* se reflejan como un arco punteado.

4.2.1 Matriz de Composición de Tipos

A la hora de generar eventos compuestos según se explica en secciones anteriores, resulta útil definir un mapa que permita visualizar se forma simple estos eventos y los tipos de casos y tipos de eventos que los producen. Para ello se propone la construcción de una matriz cuadrada en donde cada fila representa un evento compuesto asociado a un tipo caso, y cada columna representa los distintos tipos de eventos atómicos que pueden ser utilizados para la composición, cada celda puede adoptar los valores "+", "*", "-" y "!" que representa un tipo de evento atómico. En la tabla

	composite event	User	Article	Tag	ArticlesTags
User	*createUser*	+			
	updateUser	*			
	deleteUser	-			
	writesArticle		+		
	reviewArticle		*		
	unpublishArticle		-		
Article	*createArticle*		+		
	updateArticle		*		
	deleteArticle		-		
	articleTagged				+
	articleUntagged				-
	newTaggedArticle		+	[+]	+
Tag	*creatTag*			+	
	updateTag			*	
	deleteTag			-	
	tagLinked				+
	tagUnlinked				-
ArticlesTags	*createAT*				+
	updateAT				*
	deleteAT				-
	linkedArticle		+		+
	newTagArticle			+	+

Tabla 4.2: Ejemplo, tabla de composición de tipos de eventos.

4.2 se muestra una matriz de este estilo, por ejemplo, el evento compuesto "new-tagged-article" representa la creación de un nuevo artículo junto con una etiqueta, los corchetes se utilizan para simplificar, indican que un evento atómico puede no ser obligatorio en la composición, es decir, consideramos que el evento puede ser compuesto indistintamente de estas dos formas:

- $Article^{newTaggedArticle} = \{Article^{+}, Tag^{+}, ArticlesTag^{+}\}$
- $Article^{newTaggedArticle} = \{Article^{+}, ArticlesTag^{+}\}$

4.2.2 Composición, Cardinalidad de Eventos

Continuando con la idea anterior, en donde un evento atómico puede o no participar, esta situación puede ser generalizada añadiendo el concepto de cardinalidad. Así en el ejemplo anterior, $Tag^{[+]}$, puede tenderse como una cardinalidad $0 \ldots 1$, es decir, puede o no participar, y si lo hace, como mucho figura en la composición una única vez:

- $Tag^{+}_{0 \ldots 1}$
- Quedando la expresión de composición como $Article^{newTaggedArticle} = \{Article^{+}, Tag^{+}_{0 \ldots 1}, ArticlesTag^{+}\}$

La matriz anterior simplemente refleja de manera generalizada los distintos **tipos de eventos** que es posible componer, es decir, aplica directamente la definición 4.2 vista en secciones anteriores.

Recordemos que esta definición describe toda una tipología de eventos, y no eventos concretos del registro de eventos. Sin embargo podemos construir una matriz que refleje ambas ideas aplicando la definición 4.5 de **evento compuesto**, que a su vez utiliza la condición de correlación 4.3.

Esta definición no impone ninguna restricción en cuanto al número de eventos atómicos del mismo tipo a la hora componer. Es decir, según esta definición es posible construir un evento compuesto en donde participen dos eventos atómicos del mismo tipo de caso. Por ejemplo, en el caso de los artículos y etiquetas, el evento compuesto **newTaggedArticle** representa la actividad de creación de un artículo junto con alguna etiqueta. Por ejemplo, un artículo puede ser creado con un número indeterminado de etiquetas, dicho número resulta completamente irrelevante a la hora de componer el evento, pues solo interesa conocer si el artículo fue creado con o sin etiquetas (sean cuáles sean y cuántas sean). Así podemos definir cardinalidades de la siguiente forma:

- $0 \ldots 1$ cero o como mucho una aparición.
- 1 exactamente una aparición.
- $0 \ldots n$ cero o *n* apariciones.
- $1 \ldots n$ *n* apariciones, al menos una vez.

Aplicando esta notación es posible construir una matriz que refleje mayor información, en la tabla 4.3 se puede ver la misma tabla anterior, pero en donde se ven reflejadas las cardinalidades.

Continuando con esta idea, es posible basarse en modelo relacional para obtener unas cardinalidades "por defecto". Pues recordemos que la cardinalidad reflejada en el modelo relacional no condiciona a la cardinalidad en la composición de eventos. Por ejemplo, en una relación *1:n* entre dos tablas, nada nos impide definir una cardinalidad *0:1* para la composición de eventos asociados a ambas tablas. Esto queda mejor ejemplificado en el caso de los eventos de actualización (y lectura dependiendo de la situación); la cardinalidad depende completamente del domino del problema. A continuación, se describe unas pequeñas reglas generales de correspondencia entre cardinalidades del modelo relacional y la composición de eventos:

- *hasMany*, relación uno a muchos. Para eventos de creación y eliminación puede ser mapeada directamente a una cardinalidad $0 \ldots n$ o $1 \ldots n$, en el caso de eventos de lectura o modificación puede usarse cualquiera dependiendo del dominio del problema, por ejemplo $0 \ldots 1$.
- *hasOne*, relación uno a uno. Para eventos de creación y eliminación puede ser mapeada a una cardinalidad $0 \ldots 1$ o 1, en el caso de eventos de lectura o modificación puede usarse cualquiera dependiendo del dominio del problema, por ejemplo $0 \ldots 1$.

4.2.3 Guía de Composición

Como se comentó, la manera de relacionar eventos con instancias de ejecución de un proceso depende por completo del dominio del problema. Por ejemplo, si dos (o más) eventos ocurren dentro de una ventana temporal entonces es posible considerar que están relacionados a un mismo caso. Sin embargo, en este apartado se ofrece una guía práctica para la situación particular

	composite event	User	Article	Tag	ArticlesTags
User	*createUser*	[+, 1]			
	updateUser	[*, 1]			
	deleteUser	[-, 1]			
	writesArticle		[+, 1]		
	reviewArticle		[*, 1]		
	unpublishArticle		[-, 1]		
Article	*createArticle*		[+, 1]		
	updateArticle		[*, 1]		
	deleteArticle		[-, 1]		
	articleTagged				[+, 1]
	articleUntagged				[-, 1]
	newTaggedArticle		[+, 1]	[+, 0..1]	[+, 1..n]
Tag	*creatTag*			[+, 1]	
	updateTag			[*, 1]	
	deleteTag			[-, 1]	
	tagLinked				[+, 1]
	tagUnlinked				[-, 1]
ArticlesTags	*createAT*				[+, 1]
	updateAT				[*, 1]
	deleteAT				[-, 1]
	linkedArticle		[+, 1]		[+, 1]
	newTagArticle			[+, 1]	[+, 1]

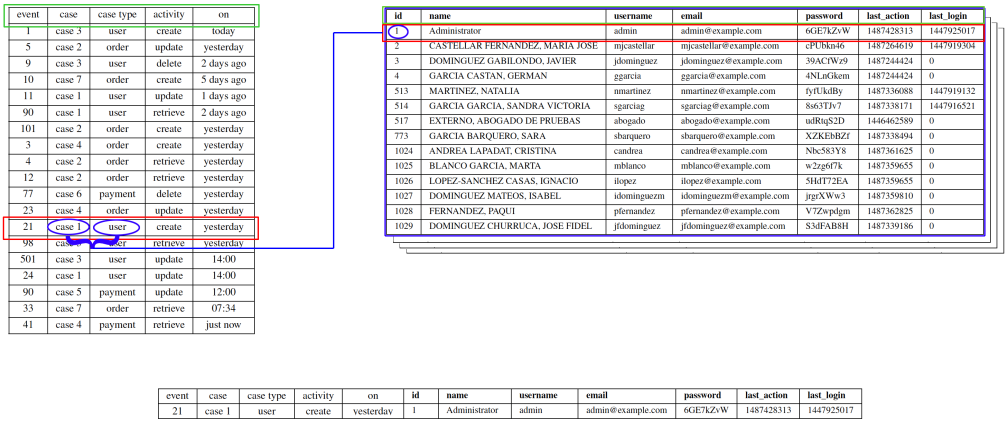
Tabla 4.3: Ejemplo, tabla de composición de eventos con cardinalidad.

sobre la que versa esta sección, es decir, cuando contamos con un modelo relacional y un sistema gestor de bases de datos que soporta a dicho modelo.

PRIMERO. Partimos de el hecho de que un evento atómico referencia siempre a registros de alguna tabla. Luego es posible desnormalizar los eventos extendiendo su conjunto de atributos, de forma que los atributos (y valores) del registro referenciado (caso) formen parte del propio evento. En Figura 4.6 se ilustra esta situación; el tipo de caso indica de cuál tabla procede el caso referenciado por el evento y el identificador del caso indica qué registro dentro de la tabla, una vez localizado, se "combina" el evento junto con la información del registro, generando así un evento extendido. En la práctica, y en este escenario, esta combinación de atributos se realiza aplicando uniones, **JOINs**, entre el registro de cambios y las tablas de donde proceden los casos que indican los eventos dentro del registro de cambios. Por otro lado, se debe prestar especial atención en el caso de eventos de eliminación, a no ser que se emplee algún mecanismo como *softdelete* o similar, es imposible extender un evento de eliminación de esta forma. Al aplicar la extensión anterior, lo que se obtiene es un **registro de eventos atómico extendido**, que ha de entenderse como una colección de elementos desnormalizados, pues carecen de una estructura común.

SEGUNDO. A continuación, es necesario identificar las claves primarias y foráneas de cada evento extendido. Esto se utilizará para correlacionar eventos con instancias de procesos.

TERCERO. Agrupar eventos extendidos por identificador de caso, esto es simplemente aplicar un predicado de correlación atómico basado en atributos, es importante conocer qué atributos referencian a un mismo caso, para ello es necesario conocer el **tipo caso**, pues según tipo se



event	case	case type	activity	on
1	case 3	user	create	today
5	case 2	order	update	yesterday
9	case 3	user	delete	2 days ago
10	case 7	order	create	5 days ago
11	case 1	user	update	1 days ago
90	case 1	user	retrieve	2 days ago
101	case 2	order	create	yesterday
3	case 4	order	create	yesterday
4	case 2	order	retrieve	yesterday
12	case 2	order	retrieve	yesterday
77	case 6	payment	delete	yesterday
23	case 4	order	update	yesterday
21	case 1	user	create	yesterday
98	case 5	user	retrieve	yesterday
501	case 3	user	update	14:00
24	case 1	user	update	14:00
90	case 5	payment	update	12:00
33	case 7	order	retrieve	07:34
41	case 4	payment	retrieve	just now

id	name	username	email	password	last_action	last_login
1	Administrator	admin	admin@example.com	6GE7kZvW	1487428313	1447925017
2	CASTELLAR FERNANDEZ, MARIA JOSE	mjcastellar	mjcastellar@example.com	cPUbkn46	1487264619	1447919304
3	DOMINGUEZ GABILONDO, JAVIER	jdominguez	jdominguez@example.com	39ACfWz9	1487244424	0
4	GARCIA CASTAN, GERMAN	ggarcia	ggarcia@example.com	4NLnGkem	1487244424	0
513	MARTINEZ, NATALIA	nmartinez	nmartinez@example.com	fyfUkdBy	1487336088	1447919132
514	GARCIA GARCIA, SANDRA VICTORIA	sgarciag	sgarciag@example.com	8s63TJv7	1487338171	1447916521
517	EXTERNO, ABOGADO DE PRUEBAS	abogado	abogado@example.com	udRtqS2D	1446462589	0
773	GARCIA BARQUERO, SARA	sbarquero	sbarquero@example.com	XZKEbBZf	1487338494	0
1024	ANDREA LAPADAT, CRISTINA	candrea	candrea@example.com	Nbc583Y8	1487361625	0
1025	BLANCO GARCIA, MARTA	mblanco	mblanco@example.com	w2zg6f7k	1487359655	0
1026	LOPEZ-SANCHEZ CASAS, IGNACIO	ilopez	ilopez@example.com	5HdT72EA	1487359655	0
1027	DOMINGUEZ MATEOS, ISABEL	idominguezm	idominguezm@example.com	jrgrXWw3	1487359810	0
1028	FERNANDEZ, PAQUI	pfernandez	pfernandez@example.com	V7Zwpdgm	1487362825	0
1029	DOMINGUEZ CHURRUCA, JOSE FIDEL	jfdominguez	jfdominguez@example.com	S3dFAB8H	1487339186	0

event	case	case type	activity	on	id	name	username	email	password	last_action	last_login
21	case 1	user	create	yesterday	1	Administrator	admin	admin@example.com	6GE7kZvW	1487428313	1447925017

Figura 4.6: Esquema de extensión de evento.
Se construye un evento extendido, en donde los atributos y valores del registro referenciado por el evento pasan a formar parte del mismo.

escogen los atributos de forma apropiada.

Por ejemplo, $Article^{articleTagged} = \{ArticlesTags^{[+,1]}\}$ (evento que ocurre cada vez que un artículo es etiquetado) se lee como: un evento de creación en **ArticlesTags** referencia a un caso en **Article**, tal evento generará un nuevo evento denominado **articleTagged** asociado al caso referenciado. Es decir, en este ejemplo concreto, queremos agrupar eventos que pertenezcan o referencien a un mismo *Article*, como en este ejemplo se trata de una composición simple, de un único evento, solo se extrae el identificador de caso (Article) a partir de cada evento de creación en ArticlesTags, es decir debemos agrupar según los valores de **ArticlesTags.article_id**, que es la clave foránea que referencia a casos de *Article*.

Otro ejemplo, $Article^{articleTagged} = \{Article^{[+,1]}ArticlesTags^{[+,0..n]}\}$, aquí debemos agrupar por **Article.id** y **ArticlesTags.article_id**, pues son estos atributos lo que identifican a casos en *Article*

Nótese que en esta fase de agrupamiento las cardinalidades de composición no juegan un papel importante en el proceso.

CUARTO. Componer, esto se traduce en la práctica en **crear nuevos eventos** en el registro de eventos (registro de cambios en la práctica) a partir de las agrupaciones anteriores. Como nombre de actividad para este nuevo evento se escoge uno arbitrario elegido en función del problema, continuando con el ejemplo anterior, como nombre de actividad se escogería *articleTagged*. Por último, en esta ocasión sí que se utilizan las cardinalidades de composición, éstas indican cuantos eventos de cada tipo se utilizan de cada agrupación (del paso tercero) para componer el nuevo evento.

CAPÍTULO 5

Aplicación Práctica

Generalmente a la hora de abordar un proyecto de minería de procesos, tanto el sector de la empresa como el tipo de procesos a analizar tiene un bajo impacto en cómo ejecutar un proyecto de esta índole. Pues sean cuales sean, el enfoque y pasos a seguir son siempre los mismos; completamente independiente de si, por ejemplo, la empresa es de servicios, se dedica a la manufactura o pertenece a un sector determinado como automoción, salud, educación, etc.

Por todo lo anterior es que en este apartado no se ofrece un *Framework* de trabajo, técnica o metodología generalista para el sector de la organización foco de este estudio, *BDA*, y lo único que se persigue es aplicar con conceptos e ideas expuestos en capítulos anteriores en un escenario real. Aunque bien es cierto que dependiendo del caso sea posible recomendar alguna técnica en particular para procesos concretos, como puede ser la aplicación de algoritmos de agrupamiento (*Clustering*) para separar y agrupar eventos del log de eventos, etc.

5.1 Sobre BDA

BDA es despacho de abogados especializado en derecho bancario y financiero. Desde el año 2015, ante el vertiginoso aumento tanto en el volumen de casos gestionados como de la propia plantilla, se pone en marcha un profundo procesos de cambio, el objetivo: informatizar y unificar al máximo posible cada aspecto de la organización. Como resultado de estos cambios nace así una herramienta de gestión unificada, a medida, para suplir todas las necesidades de gestión

que si bien en inicio su alcance se limitaba a la gestión de relaciones con clientes (CRM[1]), con el transcurrir del tiempo se ha ido escalando con la idea de ir abarcando más y más aspectos organizativos como, por ejemplo, gestión de recursos humanos o automatizaciones ligadas al ámbito financiero.

5.1.1 Modelo de Funcionamiento

El funcionamiento de esta organización es completamente tradicional y funcional, es decir, se estructura agrupando en departamentos actividades relacionadas entre sí. Su representación suele ser el organigrama, el cual establece la estructura organizativa, designa las funciones de cada trabajador y establece las relaciones jerárquicas (cadena de mando). Sin embargo, el organigrama no muestra el funcionamiento de la organización, las responsabilidades, los aspectos estratégicos, los flujos de información ni la comunicación interna. *[DomingoRey2012]*

Esta estructura tradicional, funcional o piramidal, se centra en las necesidades propias de la organización y no en las del cliente, lo cual lleva a "perder" por el camino una gran cantidad de recursos en actividades que no aportan valor, es decir, se camina hacia la ineficacia, incrementando considerablemente la burocracia, lo cual multiplica las tareas a realizar.

Esta visión departamentalizada a la larga genera diversos problemas, tales como:

- El establecimiento de objetivos locales o individuales en ocasiones incoherentes y contradictorios con lo que deberían ser los objetivos globales de la organización.
- La proliferación de actividades departamentales que no aportan valor al cliente ni a la propia organización, generando una injustificada burocratización de la gestión.
- Fallos en el intercambio de información y materiales entre los diferentes departamentos (especificaciones no definidas, actividades no estandarizadas, actividades duplicadas, indefinición de responsabilidades, etc.)
- Falta de implicación y motivación de las personas, por la separación entre "los que piensan" y "los que trabajan" y por un estilo de dirección autoritario en lugar de participativo.

El elemento principal sobre el que gira toda la organización son los de denominados *casos*, que por motivos de desambiguación desde ahora en adelante me referiré a ellos como **expedientes**. El personal de *BDA* gestiona de forma simultánea cientos de estos expedientes. Un expediente pude ser visto como un fichero físico, un archivador, una estructura sobre la que participan una serie de implicados que pueden ser personas físicas (Clientes) y/o jurídicas, por ejemplo, entidades financieras como bancos o cajas de ahorro. Un expediente viene caracterizado siempre por el "Tipo de Caso", o también conocido internamente como "producto", éstos no son más que dominaciones para los distintos servicios jurídicos ofertados por la organización; ejemplos de productos pueden ser "Preferentes", "Gastos de Hipoteca", "Cláusula Suelo", etc.

Un expediente va recopilando y generando una serie de documentación durante su ciclo de vida, documentación que al final acaba siempre siendo "empaquetada" y preparada para la última fase o estado del expediente: "presentación de la demanda", que consiste básicamente en organizar

[1] Del inglés: Customer Relationship Management

toda la documentación existente hasta el momento e imprimirla para ser presentada en los juzgados correspondientes de forma física, o bien de forma digital para presentarla en las plataformas virtuales si es posible.

Aunque en apariencia todos estos expedientes parecieran obedecer de forma estricta directrices comunes que guían al expediente durante todo su ciclo de vida, lo cierto es que dependiendo del tipo de expediente éstos pueden obedecer a flujos distintos o similares. Por ejemplo, un expediente relacionado con "Preferentes" puede ejecutar exactamente las mismas actividades que otro expediente de tipo "Cláusula Suelo", pero con un orden distinto o con una serie de condiciones añadidas. Es más, en determinadas situaciones, muy específicas, es posible que expedientes del mismo tipo no compartan el mismo flujo, es decir, alguno de los estados por los que transita un expediente pueden llegar a ser opcionales.

Ante la falta de procedimientos o diagramas de procesos en cuanto a cómo llevar la gestión de todos estos expedientes, se apicará la minería de procesos para intentar obtener un modelo lo más ajustado a la realidad de la organización. Esto es únicamente posible gracias a que el software que soporta la gestión de estas estructuras cuenta con, entre otras cosas, mecanismos de monitorización de actividad de los usuarios que la utilizan. Y aunque la actividad monitorizada es almacenada en un registro de forma estructurada y ordenada, lo cierto es que dependiendo el tipo de análisis y sobre todo las preguntas a las que se intente dar respuesta, resulta poco eficaz aplicar técnicas de minería de procesos sobre dicho registro en bruto.

5.2 Watchdog y Registro de Eventos

BDA cuenta con un CRM, una aplicación desarrollada a medida que permite principalmente la gestión integral de casos jurídicos (o también denominados "expedientes" de forma interna). Esta herramienta fue concebida desde sus inicios con idea de monitorizar al máximo posible la actividad del personal, por ello es que una de sus piezas fundamentales es el denominado *Watchdog* o "perro guardián", un elemento software que actúa a bajo nivel interceptando, observando y registrando la actividad de la base de datos. Como se observa en Figura 5.1, se trata de una especie de *Sniffer*[2] para la capa de acceso a datos, similar a lo que vendrían a ser los registros Redo[3] de Oracle.

El funcionamiento de este *Watchdog* es bastante sencillo y configurable. Aunque en principio su alcance se limita únicamente a inspeccionar colecciones o tablas del sistema de almacenamiento, en concreto, observar operaciones relacionadas con creación o modificación de datos de las colecciones que se le indican observar. Es decir, no intercepta de forma arbitraria la actividad de absolutamente cada colección ni tampoco intercepta operaciones de lectura sobre los datos del almacén; Creación, Modificación y Eliminación son las operaciones que por defecto este elemento software registra en su propio log de eventos. Por otro lado, si bien el *Watchdog* registra se forma automática las actividades antes mencionadas, éste también ofrece una interfaz que los sistemas de información pueden utilizar para registrar eventos personalizados en el log de eventos.

[2] En informática: programa que captura analiza las tramas de una red

[3] En el entorno de Oracle RDBMS, los logs Redo registra un historial de todos los cambios realizados en la base de datos

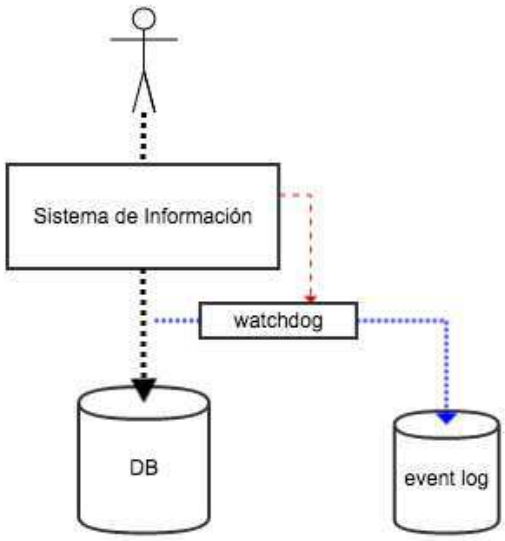


Figura 5.1: *Watchdog* o "perro guardián" del CRM de BDA.
Intercepta toda actividad sobre la base de datos principal (en azul) y registra un log describiendo en un almacén secundario. El sistema de información puede registrar logs personalizados utilizando la interfaz expuesta (en rojo) por el *Watchdog*.

En definitiva, el *Watchdog* de *BDA* resulta ser un elemento interesante desde la perspectiva de la minería de procesos, pues ofrece un registro de eventos de una calidad aceptable y ajustados a la realidad de la organización. Con una media cercana al millón de eventos registros de forma mensual hacen de este elemento software una pieza digna de estudio.

5.2.1 Características

En Figura 5.2 se puede observar un pequeño extracto del registro de *BDA* donde se pueden apreciar claras correspondencias, a continuación se describen el significado de cada una de las propiedades de este registro y sus equivalencias en un registros de eventos plano apto para la aplicación de técnicas de minería de procesos:

- *id*: Instancia de proceso.
- *user_id*: **Recurso**. Persona que ejecuta la actividad.
- *ip*: Dirección IPv4 de la persona que ejecuta la actividad.
- *(table_alias, entity_id)*: **Caso**. En concreto identifican la tabla y tupla respectivamente.
- *action*: **Actividad**.
- *extra*: Atributos adicionales, varían en función del caso y la actividad.

	A	B	C	D	E	F	G	H	I
1	id	user_id	ip	table_alias	entity_id	action	extra	created	modified
2	14650119	311	213.4.19.108	records	170112496	update	producto	24/01/2017 13:34	24/01/2017 13:34
3	14650120	112	90.74.80.180	reminders	263719	create	NULL	24/01/2017 13:34	24/01/2017 13:34
4	14650121	311	213.4.19.108	records	170112496	view	NULL	24/01/2017 13:34	24/01/2017 13:34
5	14650122	238	213.4.19.108	reminders	261367	update	NULL	24/01/2017 13:34	24/01/2017 13:34
6	14650123	118	88.26.197.41	reminders	250891	update	NULL	24/01/2017 13:34	24/01/2017 13:34
7	14650124	112	90.74.80.180	records	16070757	view	NULL	24/01/2017 13:34	24/01/2017 13:34
8	14650125	238	213.4.19.108	records	1516767	view	NULL	24/01/2017 13:34	24/01/2017 13:34
9	14650126	118	88.26.197.41	records	16010425	view	NULL	24/01/2017 13:34	24/01/2017 13:34
10	14650127	14	88.26.239.148	records	1506031	file-download	1506031/files/p	24/01/2017 13:34	24/01/2017 13:34
11	14650128	853	31.4.183.11	users	853	user-login	NULL	24/01/2017 13:34	24/01/2017 13:34
12	14650129	153	213.4.19.108	record_notes	358606	create	NULL	24/01/2017 13:34	24/01/2017 13:34
13	14650130	228	90.74.80.180	calls	1336076	delete	{	24/01/2017 13:34	24/01/2017 13:34
14	14650131	153	213.4.19.108	records	15106496	view	NULL	24/01/2017 13:34	24/01/2017 13:34
15	14650132	700	213.4.19.108	calls	1354655	create	NULL	24/01/2017 13:34	24/01/2017 13:34
16	14650133	798	79.157.244.78	records	1501425	file-moved	from:1501425	24/01/2017 13:34	24/01/2017 13:34
17	14650134	14	88.26.239.148	clients	36310	file-download	36310/files/PO	24/01/2017 13:34	24/01/2017 13:34
18	14650135	9	46.27.239.10	records	15100998	view	NULL	24/01/2017 13:34	24/01/2017 13:34
19	14650136	798	79.157.244.78	records	1501425	file-moved	from:1501425	24/01/2017 13:34	24/01/2017 13:34
20	14650137	131	88.26.239.148	reminders	263445	update	NULL	24/01/2017 13:34	24/01/2017 13:34
21	14650138	786	81.202.28.49	records	1508933	view	NULL	24/01/2017 13:34	24/01/2017 13:34
22	14650139	131	88.26.239.148	records	15106399	view	NULL	24/01/2017 13:34	24/01/2017 13:34
23	14650140	126	88.26.239.148	records	1505337	file-download	1505337/files/p	24/01/2017 13:34	24/01/2017 13:34
24	14650141	118	88.26.197.41	record_notes	358607	create	NULL	24/01/2017 13:34	24/01/2017 13:34
25	14650142	411	90.74.80.180	record_notes	358608	create	NULL	24/01/2017 13:34	24/01/2017 13:34
26	14650143	118	88.26.197.41	records	16010425	view	NULL	24/01/2017 13:34	24/01/2017 13:34
27	14650144	411	90.74.80.180	records	170114254	view	NULL	24/01/2017 13:34	24/01/2017 13:34
28	14650145	769	88.26.239.148	record_notes	358609	create	NULL	24/01/2017 13:34	24/01/2017 13:34
29	14650146	769	88.26.239.148	records	1509762	view	NULL	24/01/2017 13:34	24/01/2017 13:34
30	14650147	847	88.26.239.148	records	1408932	file-upload	1408932/files/c	24/01/2017 13:34	24/01/2017 13:34
31	14650148	258	213.4.19.108	records	170112532	view	NULL	24/01/2017 13:34	24/01/2017 13:34
32									

Figura 5.2: Extracto real registro de eventos del CRM de BDA.
Se pueden apreciar correspondencias claras entre instancias de proceso, casos, recursos y actividades.

- *created*: **Marca de Temporal**. Indica el momento en que se inicia la actividad, por lo general indica el instante en que termina de ejecutarse la actividad.
- *modificación*: Marca de Temporal, en caso de que la instancia haya sido modificada a posteriori. Por ejemplo, para indicar el término de la actividad.

Al momento de realizar este estudio el registro generado por el *Watchdog* alberga hasta la fecha un total de más de **14 millones de entradas registradas**, con un volumen de **cerca de 2GB**, y con una media de aproximadamente **1 millón de nuevas entradas registradas mensualmente**.

5.3 Watchdog como Registro de Eventos Atómico

Como se comentaba anteriormente, el *Watchdog* registra por defecto las actividades creación, modificación y eliminación. Pero cuenta con una interfaz que permite a los desarrolladores registrar eventos de forma arbitraria, por los que es posible encontrar más eventos que solo los atómicos. En la tabla 5.1 se puede observar de forma resumida para cada tabla de la base de datos los distintos tipos de eventos existentes junto con su número de apariciones. Por ejemplo, para la tabla "records" se han registrado un total de 61455 eventos de creación ("create") y 0 eventos de eliminación, luego se posible deducir que existen en la tabla un total de 61455 entradas.

Repasando el capítulo anterior, un registro de eventos atómico debe cumplir con determinadas características:

1. Ha de reflejar el **tipo de caso**.
2. Se debe poder identificar cada evento dentro del registro.

El registro del *Wathdog* cumple con ambas condiciones. El tipo de caso queda reflejado por el atributo *table_alias*, mientras que cada evento queda identificado de forma única por un identificador, el atributo *id*. Por lo cual no es necesario ningún tratamiento previo del registro, en la siguiente sección se estudiará la forma de obtener el siguiente elemento imprescindible para la obtención de eventos compuestos: un grafo de correlaciones.

5.4 Modelo Relacional y Grafo de Correlaciones

El *Watchdog* cumple con todas las características necesarias para aplicar las definiciones del capítulo anterior. El siguiente paso es obtener un grafo de correlaciones. que permita relacionar los distintos tipos de casos y de este modo poder construir eventos compuestos. En este contexto se considera a cada tabla como un tipo de caso.

5.4.1 Modelo Relacional

El modelo relacional de *BDA* que se utilizará en esta sección es el que soporta la aplicación CRM diseñada a medida por la organización. Si bien dicho modelo contiene información detallada

TABLE	*ACTIVITIES*
appointments	create (58271), delete (12845), update (116)
calls	create (116036), delete (372), update (235083)
clients	create (70582), file-download (18813), file-moved (17), file-renamed (2), file-upload (868), update (202304)
clients_records	create (98090), delete (6739)
documents	create (105556), update (10271)
hashtags	create (8625), delete (3638)
lexnet_job_task_fields	create (257931), update (107644)
lexnet_job_tasks	create (69968), update (69056)
lexnet_jobs	create (1616)
meetings	create (27036), delete (235)
orders	create (24), update (9)
payment_orders	create (43000), location-changed (90276), update (260078)
record_notes	create (364024)
records	create (61455), document-generated (130225), file-download (2898911), file-moved (237160), file-renamed (218853), file-upload (1326510), happening-changed (695), initial-payment (31899), status-changed (384183), tag-deleted (79679), update (1052530), view (4526348)
reminders	create (269931), update (383177)
room_calendars	create (14386), delete (12445)
rooms	update (3665)
storage_resources	create (62398), update (13625)
storage_resources_revisions	create (4665), delete (165), update (1)
storage_shares	create (2626), delete (217), update (71)
tags_records	create (269019), delete (79679)
users	create (573), update (21307), user-login (332783), user-logout (134192)
users_bookmarks	create (4361), delete (2265), update (187)

Tabla 5.1: Tablas observadas por el Watchdog y su conjunto de actividades registrados, entre paréntesis en número ocurrencias de cada actividad.

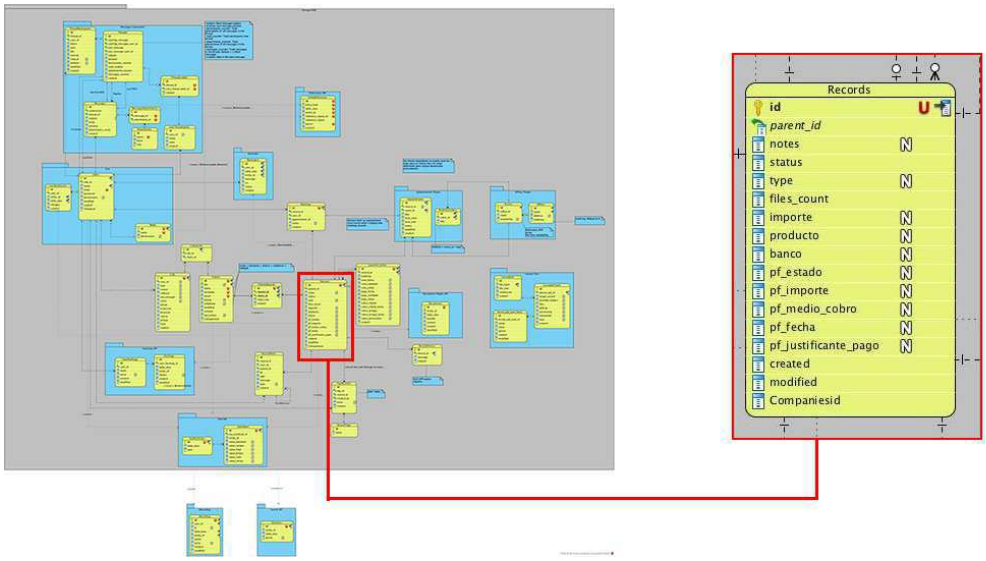


Figura 5.3: Modelo relacional físico, detalle entidad "Records".
La entidad "Records" representan los casos/expedientes sobre los que giran todos los procesos de negocio.

sobre cada tabla utilizada por el sistema, este estudio práctico se centrará en una pequeña porción del modelo, en concreto, sobre las entidades que representan los expedientes jurídicos. Pues toda la lógica del negocio gira entorno a estos elementos, en Figura 5.4 se ofrece una versión simplificada del modelo relacional centrado en estas entidades, los atributos de las entidades reflejadas en la imagen pueden no reflejar la situación real del sistema, y es que para efectos del estudio dicha información resulta completamente irrelevante.

A continuación, se describen brevemente las relaciones de interés para este estudio como se observan en Figura 5.4:

- **PhoneCall**: Registro de llamadas telefónicas. Una llamada no necesariamente debe estar relacionada con un cliente.
- **Client**: Representa a un cliente/afectado, puede ser una entidad física o jurídica.
- **Tag**: Representa un etiqueta.
- **Payment**: Representa una orden de cobro.
- **Record**: Representa a los expedientes, casos jurídicos.
- **Document**: Representa un documento, internamente es una estructura no normalizada.
- **Note**: Comentarios o notas dejados por los usuarios del sistema referidos a un expediente.
- **RecordType**: Representación del concepto de "productos" o servicios ofertados por la organización.
- **Meeting**: Representa una cita física entre un cliente y la organización para aclarar detalles sobre su expediente.

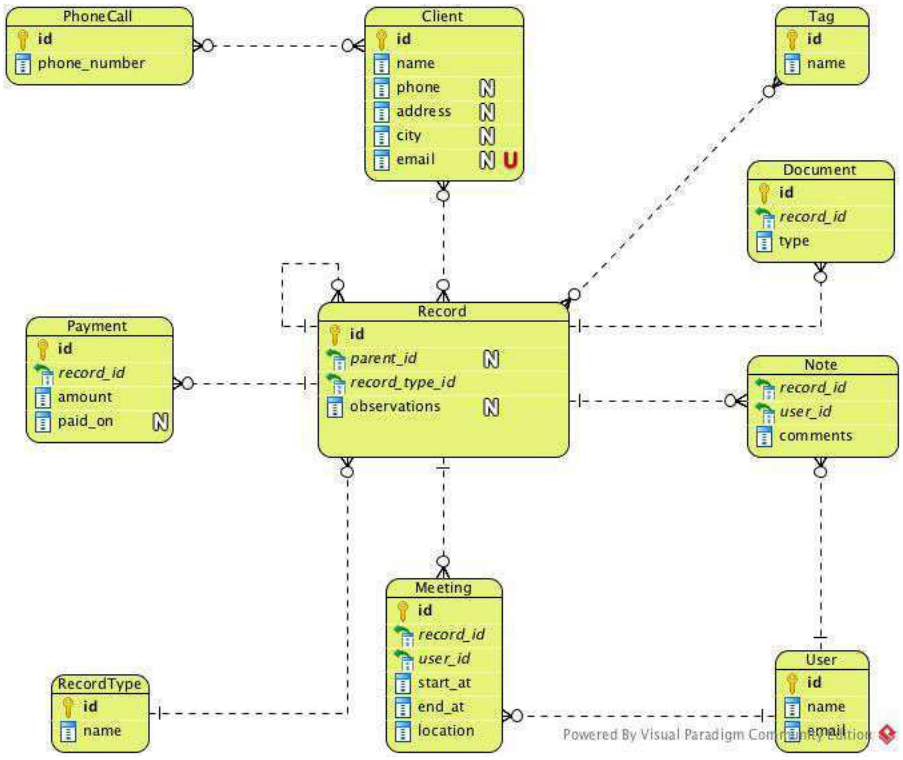


Figura 5.4: Modelo Relacional resumido.
Algunas entidades, nombres relaciones y atributos poco relevantes para el estudio han sido omitidos.

- **User**: Representa a los usuarios de la plataforma.

5.4.2 Grafo de Correlaciones

El siguiente paso es convertir el modelo relacional en un grafo de correlaciones como se describe en el capítulo anterior, utilizando como base el modelo relacional en Figura 5.4 y siguiendo las ideas del capítulo anterior, es posible construir un grafo como el que puede observar en Figura 5.5.

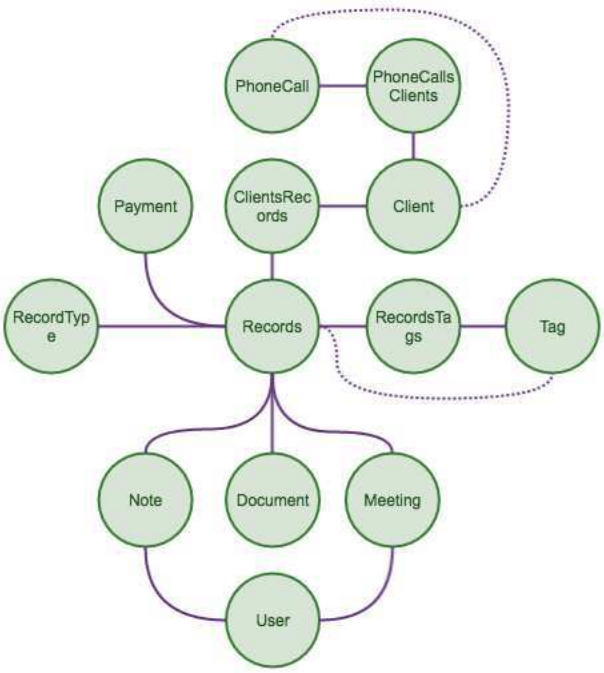


Figura 5.5: Grafo de correlaciones obtenido a partir del modelo relacional.

5.5 Matriz de Composición de Eventos

Aunque es posible calcular de forma automática todos los posibles eventos compuestos, el foco de esta aplicación práctica son los expedientes jurídicos, luego resulta innecesario realizar tal automatismo. En cambio, sí que dicho automatismo resulte de utilidad para composiciones centradas únicamente en "Records", para ello se han de considerar unicamente los tipos de eventos atómicos procedentes de tipos de casos directamente relacionados con "Records", en Figura 5.6 se pueden observar estos tipos de casos.

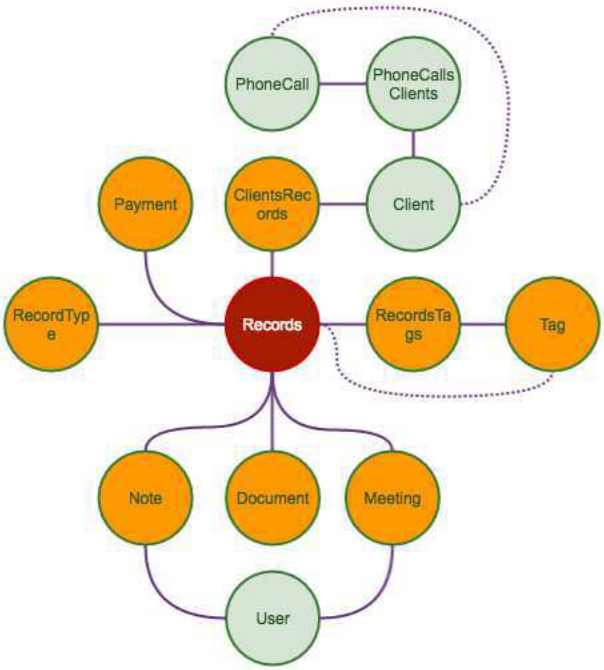


Figura 5.6: Grafo de correlaciones para tipos de casos “Records”.
En color anaranjado los tipos de casos que guardan relación directa con “Records”.

Es decir, un total de 9 tipos de casos involucrados ("Records" inclusive). Si suponemos que para cada tipo de caso existen tres tipos de eventos (create, update, delete), y deseamos crear eventos compuestos de dos dos eventos atómicos, entonces existen en principio un total de 351 eventos compuestos posibles, es decir, combinaciones de 27 (n) elementos cogidos de dos en dos (r) sin repeticiones, según dicta la ecuación $C(n,r) = \frac{n!}{r!(n-r)!}$.

De esta combinación habría que excluir algunos elementos, por ejemplo:

- Aquellos compuestos unicamente por elementos de un mismo tipo de caso.
- Aquellos que contengan un tipo de evento atómico inexistente en el registro de eventos.

Por último, como regla general, **un evento compuesto tal que todos sus eventos proceden de tipos de casos no correlacionados entre sí por lo general aportan poca información**. Por ejemplo un evento compuesto que involucre únicamente eventos atómicos de los tipos "Document" y "Meeting" no aportan información a casos de tipo "Records", a pesar que cada evento de forma individual sí que aporta información por estar directamente correlacionado con "Records". Por ello, es posible restringir el predicado de correlación visto en secciones anteriores (expresión 4.3), de forma que se exija además que todos los eventos que participan en la composición estén correlacionados entre ellos en función del tipo de caso del que proceden:

$$\boxed{V_{\psi}^{n}(v_1, v_2) = N_{\psi}(\beta(v_1), \beta(v_2)) \wedge N_{\psi}(n, \beta(v_1)) \wedge N_{\psi}(n, \beta(v_2)) \wedge P_{\psi}(v_1, v_2)} \tag{5.1}$$

Nótese la adición de la condición $N_{\psi}(\beta(v_1), \beta(v_2))$, que exige que los tipos de casos de los que proceden v_1 y v_2 estén correlacionados, es decir, exista un arco en el grafo de correlaciones entre los nodos $\beta(v_1)$ y $\beta(v_2)$.

Otra posible restricción es exigir que en la composición participe siempre al menos un evento procedente del tipo de caso objetivo (n). Por ejemplo, un evento compuesto por tres eventos atómicos, procedentes de "Document", "Meeting" y "Records", sí que puede aportar información de interés a un evento compuesto ligado a casos de "Records", por ejemplo: $Records^{v} = \{Records^{+} Meeting^{*} Document^{-}\}$.

La restricción a aplicar, o en definitiva, la elección de los eventos atómicos durante el proceso de composición dependen exclusivamente del dominio del problema. La estrategia aquí expuesta debe verse como una mera guía general de ayuda para el proceso de composición.

Aplicando estas simples reglas de purgado se puede obtener una matriz de composición como la que se puede observar en a tabla 5.2. A pesar que que interesan únicamente la composición centrada en "Records", se han incluido todos los tipos de casos con sus respectivos eventos atómicos en la diagonal que representan en cierto como la "identidad".

	composite event	PhoneCall	Client	Tag	Payment	Records	Document	Note	RecordType	Meeting	User	RecordsTags	ClientsRecords	ClientsPhoneCalls
PhoneCall	*create-phone-call*	[+, 1]												
	update-phone-call	[*, 1]												
	delete-phone-call	[-, 1]												
Client	*create-client*		[+, 1]											
	update-client		[*, 1]											
	delete-client		[-, 1]											
Tag	*create-tag*			[+, 1]										
	update-tag			[*, 1]										
	delete-tag			[-, 1]										
Payment	*create-payment*				[+, 1]									
	update-payment				[*, 1]									
	delete-payment				[-, 1]									
Records	*create-record*					[+, 1]								
	update-record					[*, 1]								
	delete-record					[-, 1]								
	tagged			[+, 0..n]								[+, 1..n]		
	untagged			[-, 0..n]								[-, 1..n]		
	client-added		[+, 0..1]										[+, 1]	
	client-removed		[-, 0..1]										[-, 1]	
	document-created						[+, 1]							
	document-removed						[*, 1]							
	document-changed						[-, 1]							
	child-added					[+, 0..n]								
	child-removed					[-, 0..n]								
	note-added							[+, 1]						
	type-changed								[*, 1]					
	meeting-planned									[+, 1]				
	meeting-changed									[*, 1]				
	meeting-cancelled									[-, 1]				
Document	*create-document*						[+, 1]							
	update-document						[*, 1]							
	delete-document						[-, 1]							
Note	*create-note*							[+, 1]						
	update-note							[*, 1]						
	delete-note							[-, 1]						
RecordType	*create-record-type*								[+, 1]					
	update-record-type								[*, 1]					
	delete-record-type								[-, 1]					
Meeting	*create-meeting*									[+, 1]				
	update-meeting									[*, 1]				
	delete-meeting									[-, 1]				
User	*create-user*										[+, 1]			
	update-user										[*, 1]			
	delete-user										[-, 1]			
RecordsTags	*create-records-tags*											[+, 1]		
	update-records-tags											[*, 1]		
	delete-records-tags											[-, 1]		
ClientsRecords	*create-clients-records*												[+, 1]	
	update-clients-records												[*, 1]	
	delete-clients-records												[-, 1]	
ClientsPhoneCalls	*create-clients-phone-calls*													[+, 1]
	update-clients-phone-calls													[*, 1]
	delete-clients-phone-calls													[-, 1]

Tabla 5.2: Matriz de composición de eventos.

5.6 Modelo de Proceso

Por último, una vez se cuenta con una matriz de composición de eventos el siguiente paso consiste en analizar el registro de eventos disponible, truncar la información a la únicamente imprescindible y aplicar los algoritmos de minería de procesos.

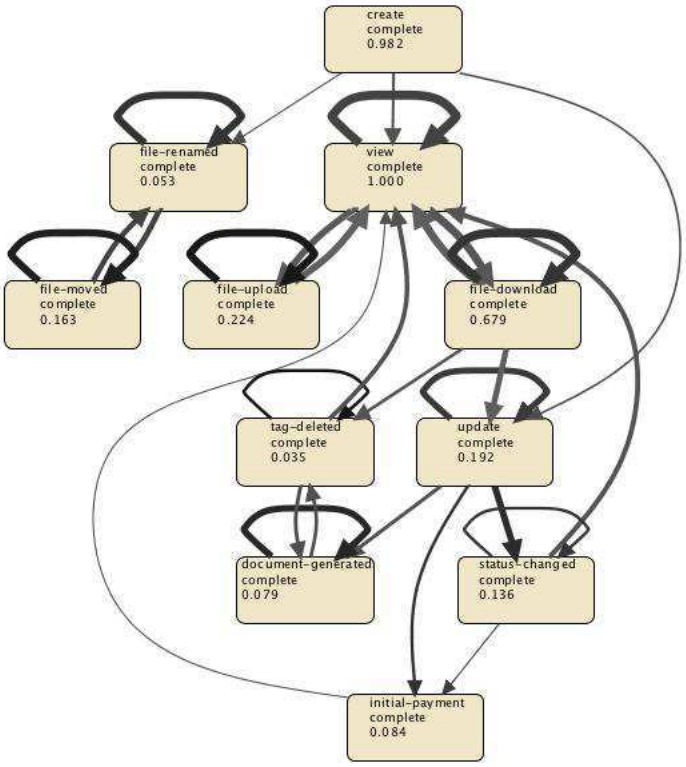


Figura 5.7: Modelo preliminar descubierto utilizando el algoritmo *Fuzzy Miner* con software *ProM 6*.

Modelo de proceso descubierto sin filtrar ni estudiar los eventos disponibles, resulta bastante trivial y aporta poca información o utilidad práctica.

En esta sección no se entrará en detalles sobre cómo pre-procesar y purgar el registro de eventos, ni en cómo se construyen en la práctica los eventos compuestos. Recordemos que la principal problemática a la hora de construir trazas de ejecución radica en cómo correlacionar eventos con instancias (casos), solo se mencionará que conociendo el modelo relacional y los esquemas de cada tabla es posible correlacionar eventos con casos casi de una manera trivial siguiendo la "Guía de Composición" de la sección anterior.

Recordemos una vez más, el objetivo de este capítulo es aplicar la minería de procesos para obtener un modelo sobre la gestión de los expedientes jurídicos de *BDA*. En Figura 5.7 se puede observar un modelo de proceso obtenido directamente sobre el registro de eventos producido

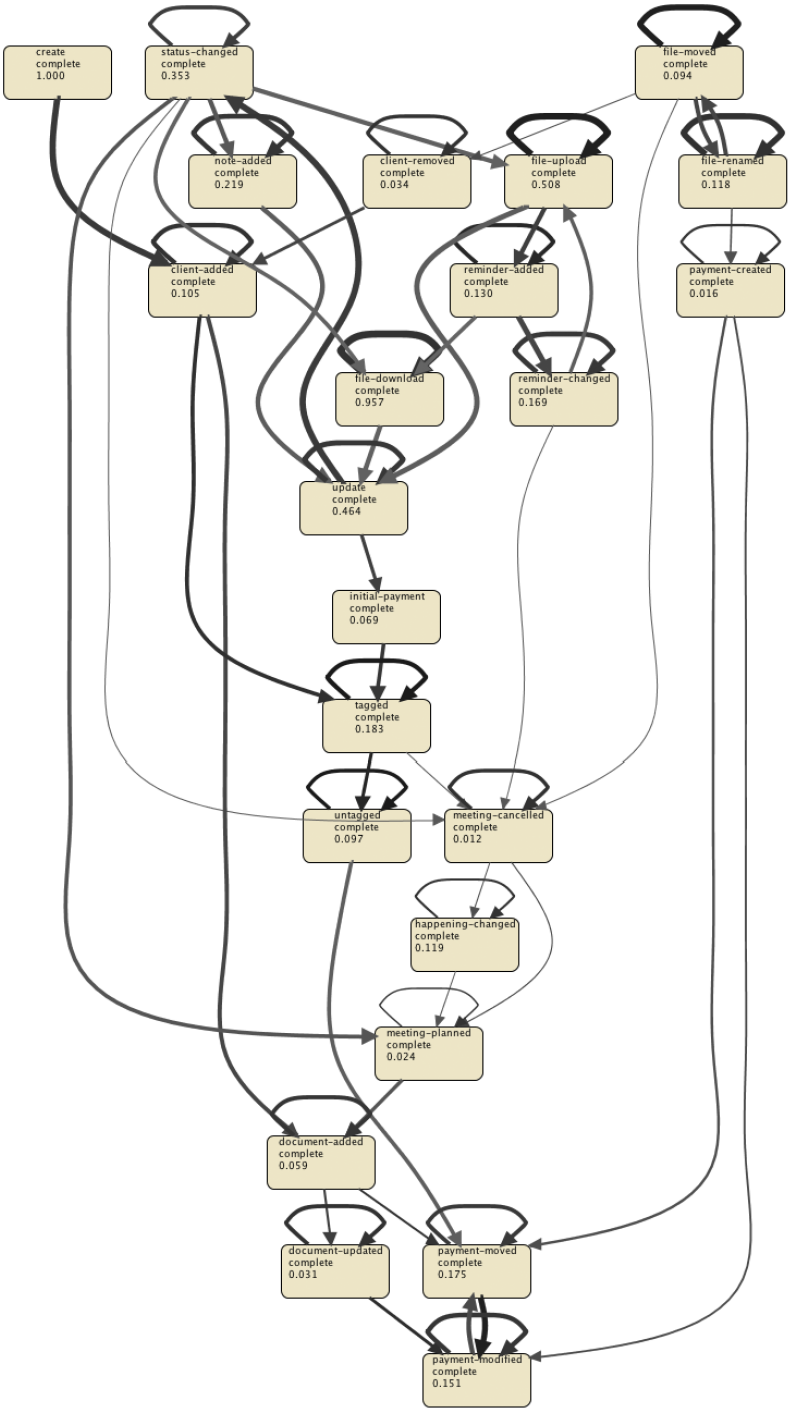


Figura 5.8: Modelo de proceso descubierto aplicando la composición de eventos.
Modelo obtenido siguiendo la matriz de composición anterior, el grosor de los arcos representa la frecuencia con la que ocurre cada actividad; a mayor grosor mayor frecuencia.

por el *Watchdog*, la información que aporta resulta de poca utilidad y en algunos casos parece incompleta. Por ejemplo, se observa la presencia de la actividad *tag-deleted* que ocurre cada vez que un expediente es etiquetado, pero no se observa su contraparte *tag-added* que refleja el caso contrario, esto puede ocurrir por inconsistencias en el software, recordemos que el *Watchdog* ofrece una API que permite a los desarrolladores registrar de forma manual un evento. Por lo tanto, la falta de dicha actividad se debe a que los desarrolladores registran manualmente la actividad *tag-deleted* en cada expediente cada vez que ocurre una eliminación en la tabla intermedia, pero han olvidado contemplar el caso contrario, esta hipótesis queda además respaldada por el hecho de que el número de eventos *tag- deleted* coincide exactamente con el número de eventos de eliminación sobre la tabla intermedia *tags_records* (79.679).

5.6.1 Comparativa Por Tipo de Expediente

El modelo de procesos anterior ofrece un visión un poco más aproximada a la realidad, aunque, conociendo la lógica de negocio de la organización la intuición dicta que expedientes jurídicos de distinta tipología deberían tener comportamientos distintos aunque compartan un conjunto de actividades común. Por ejemplo, el orden de ejecución de las actividades según el tipo de expediente puede resultar interesante de estudiar. En Figura 5.9 se pueden observar el número de expedientes según su tipo, en este apartado se abordarán los tres primero con mayor volumen, se obtendrá un modelo de procesos por cada uno de ellos y se determinará si poseen diferencias significativas.

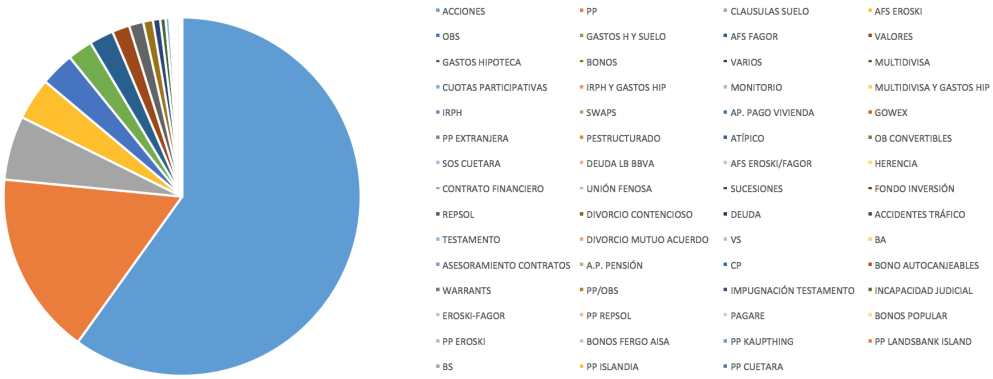


Figura 5.9: Número de expedientes por tipo.
Los tres tipos de expedientes (productos) con mayor volumen son “ACCIONES”, “PP” y “CLAUSULAS SUELO”

Una inspección superficial revela claras diferencias entre los modelos, tanto en las frecuencias de algunas actividades como en el orden de ejecución de las mismas. Otro aspecto a comparar es el paralelismo, o concurrencia, entre actividades; que son ejecutadas de forma concurrente, es decir, en el mismo instante de tiempo.

Por otro lado, se logra apreciar que el flujo habitual suele ser: Alta de expediente, se añaden clientes/participantes al mismo y luego se planifica una cita en las oficinas de la organización

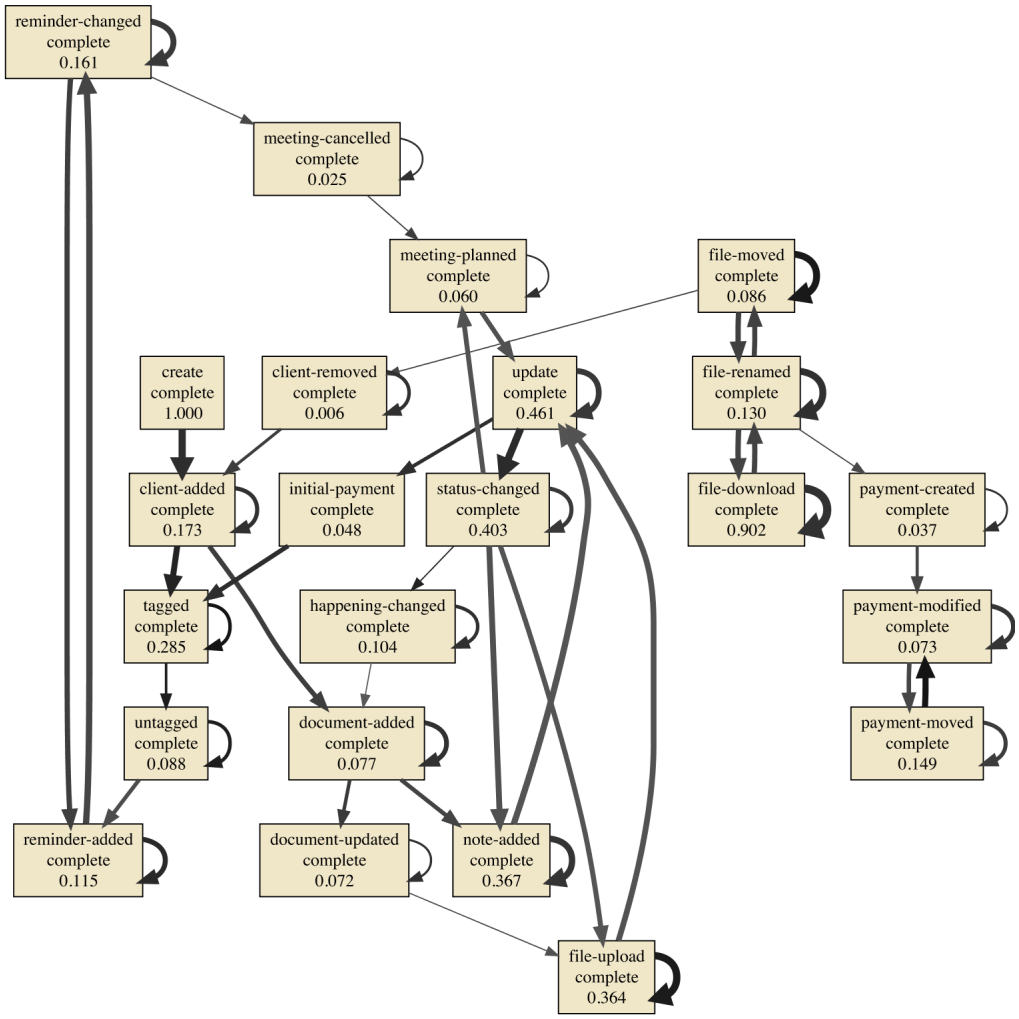


Figura 5.10: Modelo de proceso para expedientes de tipo “ACCIONES”.

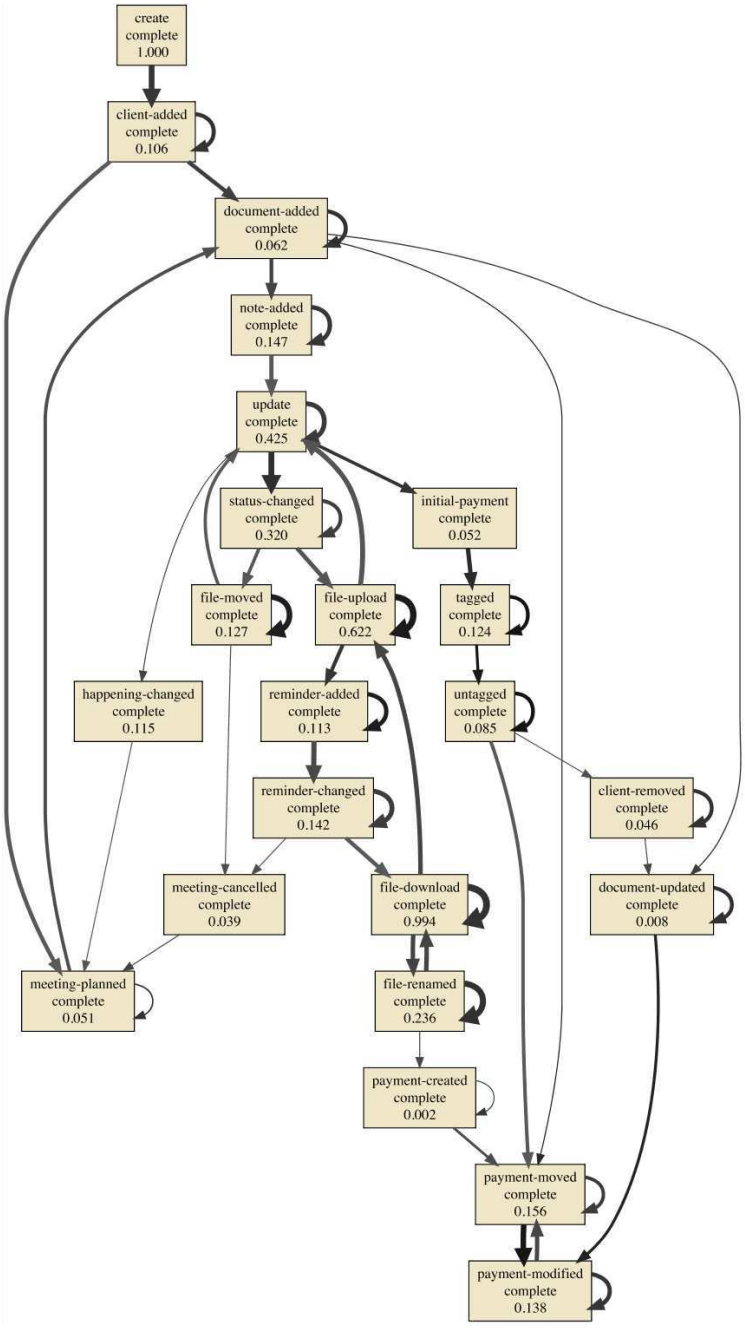


Figura 5.11: Modelo de proceso para expedientes de tipo “PP”.

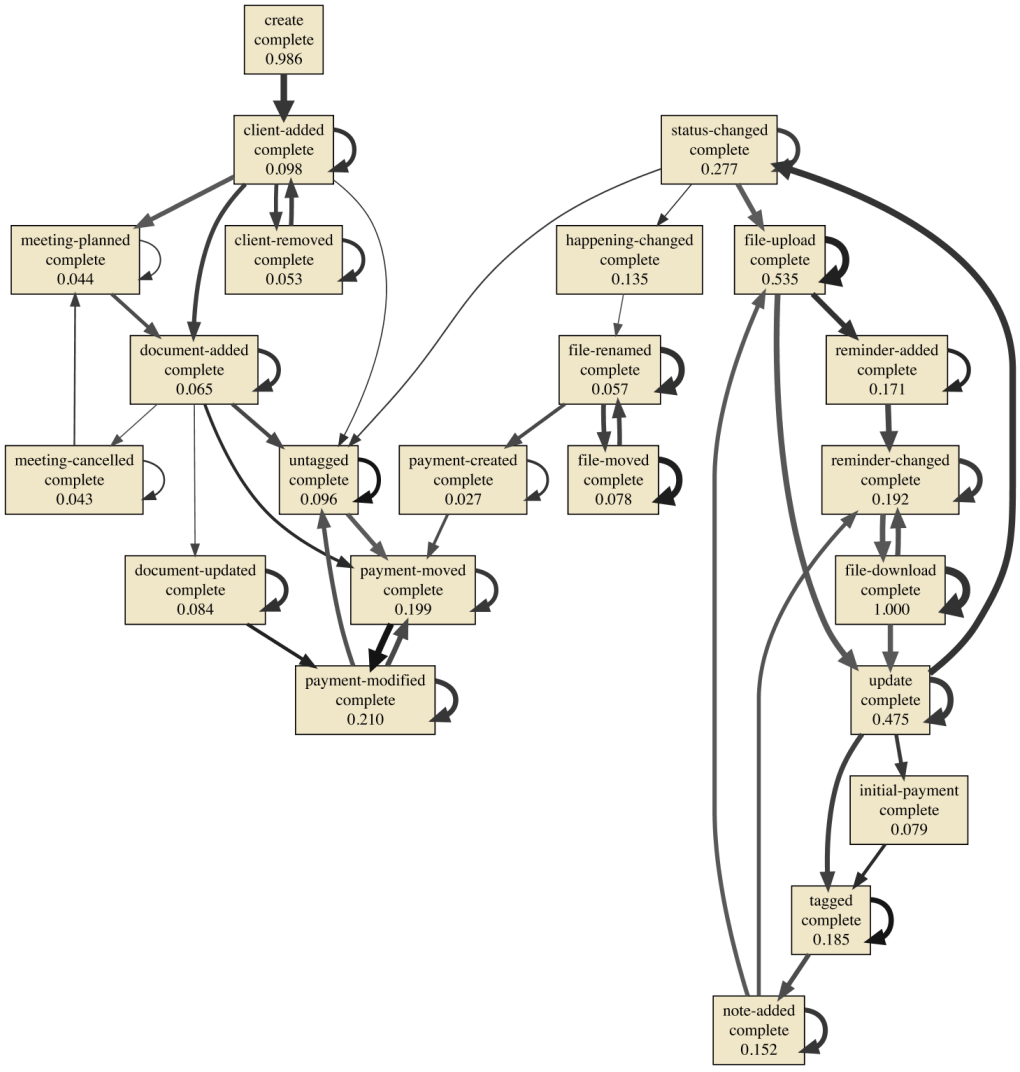


Figura 5.12: Modelo de proceso para expedientes de tipo “CLAUSULAS SUELO”.

para tratar el caso. Esto ocurre en los expedientes de tipo “PP” y “CLASULAS SUELO”, sin embargo, los expedientes de tipo “ACCIONES” parecen omitir el paso de concertar una cita con el cliente en la mayoría de casos.

CAPÍTULO 6

Conclusiones y Líneas Futuras

En este documento se han ofrecido una serie de ideas y definiciones que permiten conceptualizar la problemática de la correlación de eventos bajo condiciones concretas. En la minería de procesos, la mayor parte del tiempo no se invierte en las labores de análisis como cabría de esperar, sino en tareas de localización, filtrado y selección de los datos, eventos. Si bien es cierto que en este sentido no se ofrece un mecanismo generalista que permita automatizar parte o la totalidad de las tareas, sí que se ofrece una visión generalizada y formal de un escenario concreto. Evidentemente el principal campo de aplicación de estas ideas es aquel en donde el registro de eventos proviene de una base de datos, en donde las modificaciones de cada tabla pueden ser entendidas como eventos. En este documento se conceptualiza este punto de vista y se ofrecen estrategias que permiten, en cierto modo, crear un modelo del problema.

Con la idea de poner a prueba estos conceptos, se ha ofrecido una aplicación práctica sobre un escenario real. Aunque cabe señalar que los detalles técnicos de tal implementación se escapan al alcance de este documento, por lo que se describen de manera escueta y ofreciendo directamente sus resultados. Como dato destacable, durante la aplicación de los conceptos, la *matriz de composición de eventos* ha demostrado resultar extremadamente útil en las labores de construcción del registro de eventos, pues permite visualizar de una manera organizada y sencilla los distintos eventos que es posible construir.

A falta de herramientas para asistir en las tareas de confección de eventos mediante composición, el siguiente paso natural de este estudio sería la creación de dichas herramientas. Por ejemplo, una herramienta que dado un grafo de correlaciones y un registro de eventos atómicos generase de forma automática una matriz de composición de eventos es completamente factible y resultaría de utilidad. Sin embargo, esto requiere de en estudio en mayor profundad sobre el

asunto de qué eventos y de qué tipo en concreto se han de coger para crear un nuevo evento compuesto, pues dicha tarea es completamente asistida y dependiente del dominio del problema, sería interesante desarrollar mecanismos que permitan formalizar y automatizar esta labor de selección, por ejemplo, alguna medida de ganancia de información permitiría escoger las mejores combinaciones posibles dentro de todo el conjunto de posibles combinaciones.

Por otro lado, resultaría interesante el estudio de si la composición recursiva de eventos es posible, recordemos que en este documento solo se ha propuesto la composición de eventos únicamente a partir de eventos atómicos.

Por último, y en relación a la aplicación práctica de los conceptos. Una vez obtenidos los modelos, el siguiente paso es realizar sería una comparativa cruzada entre ellos y de algún modo determinar el grado de similitud. Sin embargo, dado que dicho análisis se escapa al alcance de este estudio, únicamente se limitará a comentar que, en efecto, existen algunos estudios recientes al respecto, como, por ejemplo, en *[Jeroen2015]* se describe una estrategia de comparación que utiliza un registro de eventos para encontrar y visualizar las diferencias entre dos o más modelos de procesos, o como en *[Azzurra2015]* que proporciona un método de comparación visual similar. Existen otros estudios, como por ejemplo *[SergeyIvanov2015]*, cuya estrategia se basa en comprar directamente modelos basado en su notación XML.

Bibliografía

[WeijtersAalst2003] Weijters, A.J.M.M., Aalst, W.M.P.V. der, 2003. Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering 151–162.

[Aalst2004] Aalst, W.M.P. van der, Weijters, A.J.M.M., Maruster, L., 2004. Workflow Mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16, 1128–1142.

[AalstReijersSong2005] Aalst, W.M.. V.D., Reijers, H.A., Song, M., 2005. Discovering social networks fromevent logs. Computer Supported Cooperative Work (CSCW) 14, 549–593

[RozinatAalst2006] Rozinat, A., Van Der Aalst, W.M.P., 2006. Decision mining in ProM, in: Proceedings, Lecture Notes in Computer Science. Presented at the 4th International Conference on Business Process Management, Springer, pp. 420–425.

[Hornix2007] Hornix, P.T.G., 2007. Performance Analysis of Business Processes through Process Mining.

[Aalst2007] W.M.P. van der Aalst et al. Business Process Mining: An Industrial Application

[HamidReza2009] Hamid Reza Motahari-Nezhad, Regis Saint-Paul, Fabio Casati, Boualem Benatallah Event correlation for process discovery from web service interaction logs

[VanAalst2011] W.M.P. Van der Aalst et al. Process mining: Discovery, conformance and enhancement of business process. Berlin: Springer.

[AalstAdriansyah2011] Aalst, W.M.P.V. der, Adriansyah, A., de, A.K.A., Medeiros, Arcieri, F., et al., 2011. Process Mining Manifesto. Business Process Management Workshops 2011, Lecture Notes in Business Information Processing. Springer-Verlag 99.

[DomingoRey2012] Domingo Rey Peteiro. Sinapsys Business Solutions, S.L. La Gestión Tradicional y la Gestión por Procesos

[Azzurra2015] Azzurra Pini, Ross Brown & Moe T. Wynn Process visualization techniques for multi-perspective process comparisons

[Gonzalez2015] E. Gonzalez López de Murillas, W.M.P van der Aalst & H.A. Reijers1 Process Mining on Databases: Unearthing Historical Data from Redo Logs

[Jeroen2015] J.A.J. (Jeroen) van Mourik Comparing Business Processes using Event Data

[SergeyIvanov2015] Sergey Ivanov & Anna Kalenkova Comparing Process Models in the BPMN 2.0 XML Format

[Aalst2015] W.M.P. van der Aalst Extracting event data from databases to unleash process mining

[VanAalst2016] W.M.P. Van der Aalst et al. Process Mining. Data Science in Action. Second Edition. Berlin: Springer.